Automates Cellulaires : Aspects algorithmiques des configurations périodiques en toute dimension

Nicolas Bacquey

UNIVERSITÉ CAEN NORMANDIE

Normandie Université

Université de Caen Normandie

École doctorale SIMEM

Thèse de doctorat

Présentée et soutenue le : 04/12/2015 par

Nicolas Bacquey

en vue de l'obtention du

Doctorat de l'Université de Caen Normandie

Spécialité : Informatique et applications

Automates Cellulaires : Aspects algorithmiques des configurations périodiques en toute dimension

Composition du jury :

Olivier Bournez	Professeur des Universités	École Polytechnique	*Rapporteur*
Bruno Durand	Professeur des Universités	Univ. Montpellier 2	*Rapporteur*
Victor Poupet	Maître de Conférences	Univ. Montpellier 2	*Rapporteur*
Valérie Berthé	Directrice de Recherche CNRS	Univ. Paris Diderot	*Examinatrice*
Éric Rémila	Professeur des Universités	Univ. Jean Monnet	*Examinateur*
Étienne Grandjean	Professeur des Universités	Univ. Caen Normandie	*Co-directeur*
Gaétan Richard	Maître de Conférences	Univ. Caen Normandie	*Co-directeur*
Véronique Terrier	Maître de Conférences	Univ. Caen Normandie	*Co-directrice*

Remerciements

Je tiens en premier lieu à exprimer ma gratitude envers mes directeurs de thèse. Merci donc à Étienne Grandjean, Véronique Terrier et Gaétan Richard pour la confiance qu'ils ont placé en moi pendant ces trois années de thèse, mais également pour leurs multiples conseils avisés. Il est évident que leur compétence, leur disponibilité à toute épreuve et leur soutien ont été des facteurs essentiels aux travaux effectués pendant cette thèse. Je leur suis reconnaissant d'avoir su me transmettre le goût de la recherche fondamentale.

Je remercie sincèrement mes relecteurs, pour le temps qu'ils m'ont consacré et leurs précieuses remarques. Merci en particulier à Olivier Bournez, Bruno Durand et Victor Poupet, qui m'ont fait l'honneur d'être mes rapporteurs, et à Valérie Berthé et Éric Rémila, qui ont accepté d'être membres de mon jury.

Une grande partie de la réussite (ou de l'échec) d'une thèse repose également sur l'environnement dans lequel elle a été effectuée. Pour cette raison, je tiens à remercier les membres du GREYC pour m'avoir accueilli dans leur laboratoire. En particulier, il me faut remercier les membres de l'équipe AmacC pour le cadre scientifique et humain exceptionnel qu'ils ont su mettre en place. Notre séminaire hebdomadaire va beaucoup me manquer. Je n'oublie pas non plus de remercier les administrateurs systèmes et le personnel administratif du GREYC : leur compétence et leur efficacité ont été d'un grand secours à de nombreuses reprises.

Je tiens à exprimer ma gratitude envers les autres thésards (ou ex-thésards) du GREYC, et particulièrement à Thibaut, Paul, Jean et José. Avoir pu mélanger la détente et la travail a été particulièrement agréable, et a été un important facteur de motivation.

Je tiens également à saluer les malheureux qui ont eu l'infortune de partager mon bureau au quotidien. Merci à Morgan et à Jean d'avoir supporté tour à tour mon mutisme et ma loquacité, mes manies et mon humeur. Je pense que nos discussions et nos digressions ont largement contribué à l'élaboration de ce manuscrit.

Enfin, je me permets d'adresser mes sentiments à ma famille. Merci à mes parents d'avoir encouragé ma curiosité, et de n'avoir jamais désespéré de répondre à mes Pourquoi et mes Comment. Merci à mes frères pour leur présence quand j'avais besoin d'eux. Finalement, je tiens tout particulièrement à remercier ma femme, pour sa patience, sa compréhension et son soutien.

Table des matières

Chapitre 1

Introduction

Le modèle des automates cellulaires a été introduit par Ulam et von Neumann [53] dans les années 40-50 autant pour des raisons théoriques que dans le but de modéliser des phénomènes complexes : auto-réplication, etc. Depuis les années 50, ce modèle parallèle et synchrone a suscité une énorme quantité de travaux touchant à la modélisation, en physique théorique, biologie, géologie, voire en sciences économiques et en sciences sociales [29, 51, 17]. On peut expliquer un tel succès d'une part par la simplicité de ce modèle, d'autre part par son uniformité et son caractère local qui en font un modèle réaliste (pour un aperçu des thématiques en informatique théorique, voir [46]).

D'un de vue point théorique, le modèle des automates cellulaires se présente sous deux aspects principaux. D'une part, il constitue l'exemple typique d'un modèle de calcul massivement parallèle, local et uniforme avec des problématiques intéressantes [48]. D'autre part, ce modèle peut être étudié comme un système dynamique discret ; de ce point de vue un automate cellulaire est une fonction continue qui commute avec le shift [21]. L'étude des automates cellulaires en tant que système dynamique est un domaine de recherche actif, dans lequel de nombreuses questions restent ouvertes. Dans cette thèse, nous prenons le parti de nous y intéresser en tant que modèle de calcul, tout en considérant des notions issues des systèmes dynamiques, comme les configurations périodiques et les points fixes.

L'objectif principal de cette thèse est d'apporter une première contribution à une algorithmique et à une théorie de la complexité des automates cellulaires agissant sur des configurations périodiques. Cet objectif est motivé par les deux points suivants :

1. Les configurations périodiques sont des objets naturels. Dans la littérature, les configurations périodiques de dimension 1, et surtout de dimension 2, ont été très étudiées, d'une part en tant que pavages [7], d'autre part dans le cadre des automates cellulaires considérés comme des systèmes dynamiques [37, 38, 23].
2. À l'inverse, nous n'avons pas trouvé de travaux antérieurs à notre article de 2014 [2] qui portent sur l'algorithmique et la complexité des automates cellulaires agissant sur des configurations périodiques. Dans la littérature, l'algorithmique et la complexité des automates cellulaires se sont construites essentiellement dans le modèle standard, c'est-à-dire, le

modèle où la configuration initiale de l'automate cellulaire est son entrée entourée de symboles spéciaux [50].

Il est clair que le modèle standard est le modèle le mieux adapté à l'algorithmique et à la complexité des automates cellulaires. Il permet de définir par analogie avec le modèle de Turing des ressources de calcul précises — le temps et l'espace — et les classes de complexité correspondantes : espace linéaire, temps linéaire (ou temps polynomial) avec espace linéaire, etc [14].

Pourtant, par certains aspects, les configurations périodiques méritent d'être étudiées aussi dans le cadre du calcul. En dimension 1, par exemple, il peut paraître aussi naturel, voire plus naturel du point de vue du modèle, de présenter la configuration initiale sur le mot d'entrée w sous la forme périodique $\dots www\dots$ plutôt que par le mot w entouré de symboles spéciaux (c'est sous cette forme qu'est présenté le problème de classification de densité [27, 19]).

Ce n'est pas tout à fait un hasard si, jusqu'à présent, une algorithmique et une théorie de la complexité n'ont pas été élaborées pour le modèle des configurations périodiques. Contrairement au modèle de Turing, ce modèle est intrinsèquement uniforme, ce qui pose des limitations, comme nous le verrons, dans le cas de la dimension 1 et pour les dimensions supérieures :

- **Des propriétés de clôture :** les objets, langages (de mots ou d'images) ou fonctions, calculés par les automates cellulaires périodiques ont nécessairement certaines propriétés de clôture : en dimension 1, il est facile de vérifier que tout langage reconnu par un automate cellulaire périodique est un langage *cyclique*, c'est-à-dire qu'il est clos par shift (permutation circulaire), puissance et racine.
- **La formalisation de l'arrêt du calcul :** du fait de la présentation périodique de l'entrée, on ne peut identifier aucune cellule spécifique pour détecter l'arrêt du calcul. L'arrêt et le résultat du calcul ne peuvent se détecter que par une condition globale, typiquement un point fixe ou un cycle limite.
- **La définition de la complexité :** comme dans le modèle standard, on doit définir la complexité (en temps ou espace) d'un algorithme en fonction de la longueur de l'entrée. Pour une configuration périodique, la référence intrinsèque ne peut être que la longueur d'une période minimale.

Prenant en compte les points précédents, les questions principales que nous étudions dans cette thèse concernant le modèle des configurations périodiques, appelé pour faire court modèle périodique, sont les suivantes :

1. Comment définir une période minimale ? Avec quel algorithme et quelle complexité ?
2. Quels sont les liens entre les classes de complexité définies dans le modèle standard et les classes de complexité du modèle périodique ?

Nous chercherons à répondre à ces deux questions pour toutes les dimensions. Alors qu'en dimension 1, nous obtenons des réponses qu'on peut juger complètes, nos résultats sont plus partiels dans les cas plus complexes de la dimension 2 et des dimensions supérieures, mais permettent d'aborder d'autres questions avec une optique nouvelle.

Dans cette thèse, nous présenterons dans un premier temps le modèle de calcul des automates cellulaires, ainsi que différents outils et concepts nécessaires à la conception d'algorithmes, tels que les signaux et les fonctions constructibles en temps. Nous mettrons en œuvre ces concepts dans un premier exemple d'algorithme sur un automate cellulaire de dimension 2.

Le chapitre suivant sera consacré à l'algorithmique des configurations périodiques en dimension 1. En particulier, nous montrerons que le modèle standard et le modèle périodique sont équivalents en ce qui concerne la reconnaissance de langages et le calcul de fonction, moyennant quelques restrictions naturelles sur la forme de ces langages et fonctions. Le principal outil de ce résultat d'équivalence est un algorithme polynomial capable d'identifier la période minimale d'une configuration périodique de dimension 1.

Ensuite, nous tenterons d'adapter les résultats d'équivalence obtenus précédemment aux automates cellulaires et aux configurations périodiques de dimension 2. Nous n'y parviendrons que partiellement, le principal écueil résidant dans le fait que le passage à la dimension supérieure induit une généralisation non triviale du concept de mot primitif. Nous caractériserons toutefois cette généralisation sous le concept de racine primitive, et présenterons un algorithme capable de calculer les racines primitives de toute configuration périodique de dimension 2. Cet algorithme est essentiellement un algorithme d'élection de leader.

Enfin, nous analyserons dans un dernier chapitre les possibilités d'étendre en dimension quelconque les résultats obtenus avec les outils que nous avons développés pour la dimension 2. Nous adapterons ces outils afin que leurs définitions restent valables en toute dimension, et transposerons la quasi totalité des résultats obtenus pour la dimension 2 en dimension quelconque. Nous présenterons notamment une version de l'algorithme d'élection de leader et de calcul de racines primitives capable d'opérer en dimension quelconque. Cette version est plus simple dans sa mise en œuvre que l'algorithme spécifique à la dimension 2, mais également plus coûteuse en temps.

Chapitre 2

Préliminaires : Concepts manipulés, outils et notations

Nous allons tenter dans ce chapitre de familiariser le lecteur avec le modèle de calcul au cœur de cette thèse, celui des automates cellulaires. En particulier, nous aborderons les notions de configurations périodiques, de diagrammes espace-temps et de signal. Nous développerons le concept de reconnaissance de langage par un automate cellulaire, notamment en précisant comment un tel modèle de calcul intrinsèquement infini peut accepter ou rejeter des mots finis.

Pour terminer, nous présenterons un algorithme sur les automates cellulaires de dimension 2 qui synthétisera les concepts évoqués tout au long du chapitre.

2.1 Automates cellulaires

2.1.1 Le modèle de calcul

Nous utiliserons dans cette thèse la définition standard d'un automate cellulaire synchrone et déterministe (voir [25]).

Définition 2.1 (automate cellulaire)**.** *Un* automate cellulaire *est un tuple* $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$ *où :*

- $d \in \mathbb{N}^*$ *est la* dimension *de l'automate cellulaire, et indique que le réseau sous-jacent est* $\mathbb{Z}^d$*.*
- $\mathcal{Q}$ *est un ensemble fini d'*états*.*
- $\mathcal{V} \subset \mathbb{Z}^d$ *est un sous-ensemble fini ordonné (appelé* voisinage*) du réseau sous-jacent* $\mathbb{Z}^d$*.*
- $\delta : \mathcal{Q}^{\mathcal{V}} \to \mathcal{Q}$ *est la* fonction de transition locale *de l'automate.*

La plupart des automates cellulaires traités dans cette thèse auront pour fonction de reconnaître des langages. Pour cette raison, on définit un alphabet d'entrée dans lequel seront exprimés les langages caractérisés par un automate cellulaire.

Définition 2.2 (alphabet d'entrée)**.** *L'*alphabet d'entrée *d'un automate cellulaire* $\mathcal{A}$ *est un sous-ensemble* $\Sigma \subseteq \mathcal{Q}$ *de son ensemble d'états.*

Définition 2.3 (configuration d'un automate cellulaire, cellule)**.** *Une* configuration *d'un automate cellulaire $\mathcal{A}$ est une fonction $\mathcal{C} : \mathbb{Z}^d \to \mathcal{Q}$ qui associe un état de l'automate à chaque point du réseau $\mathbb{Z}^d$. La* configuration initiale *d'un automate pour un calcul donné est une fonction $\mathcal{C}_{init} : \mathbb{Z}^d \to \Sigma$.*

Une cellule *de la configuration $\mathcal{C}$ d'un automate cellulaire est un élément $z \in \mathbb{Z}^d$. L'état de la cellule z dans la configuration $\mathcal{C}$ est noté $\mathcal{C}(z)$.*

Nous allons particulièrement nous intéresser dans cette thèse aux automates cellulaires agissant sur des configurations périodiques. En voici une définition formelle :

Définition 2.4 (configuration périodique)**.** *Une configuration $\mathcal{C} : \mathbb{Z}^d \to \mathcal{Q}$ est* périodique *s'il existe une famille libre de vecteurs $u_1, u_2, \ldots, u_d \in \mathbb{Z}^d$ telle que :*

$$\forall z \in \mathbb{Z}^d \ \forall i \in [\![1; d]\!] : P(z + u_i) = P(z)$$

La fonction de transition locale d'un automate cellulaire induit une fonction globale sur l'ensemble des configurations.

Définition 2.5 (fonction de transition globale)**.** *La* fonction de transition globale *d'un automate cellulaire $\mathcal{A}$ de voisinage $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ est la fonction $F_\delta : \mathcal{Q}^{\mathbb{Z}^d} \to \mathcal{Q}^{\mathbb{Z}^d}$ résultant de l'application synchrone et uniforme de la fonction de transition locale δ en tout point d'une configuration. En d'autres termes :*

$$\forall z \in \mathbb{Z}^d, \ \forall \mathcal{C} \in \mathcal{Q}^{\mathbb{Z}^d}, \ F_\delta(\mathcal{C})(z) = \delta(\mathcal{C}(z + v_1), \mathcal{C}(z + v_2), \ldots, \mathcal{C}(z + v_n))$$

Dans la suite de cette thèse, nous ne nous intéresserons qu'à un seul type de voisinage pour les automates cellulaires, nommé « voisinage de Moore ».

Définition 2.6 (voisinage de Moore)**.** *Le* voisinage de Moore *en dimension d est l'ensemble $\mathcal{V} = \{z = (z_1, \ldots, z_d) \in \mathbb{Z}^d : \max(|z_1|, \ldots, |z_d|) \leq 1\}$. Il est ordonné par l'ordre lexicographique sur $\mathbb{Z}^d$.*

Par la suite, on supposera que tous les automates cellulaires présentés utilisent le voisinage de Moore de la dimension appropriée. Les capacités de différents types de voisinages ont déjà été étudiées, notamment dans [42]. Dans le cadre de cette thèse, nous en retiendrons que tous les voisinages sont équivalents pour les complexités en temps au moins linéaire (les seules que nous considérerons), à la condition qu'ils permettent à une information de se « déplacer » n'importe où sur la configuration.

Définition 2.7 (état persistant, état quiescent)**.**

— *Un état $p \in \mathcal{Q}$ est* persistant *s'il n'est pas modifié par application de la règle de transition, quel que soit son voisinage, autrement dit si*

$$\forall \mathcal{C} \in \mathcal{Q}^{\mathbb{Z}^d} \ \forall z \in \mathbb{Z}^d \quad \mathcal{C}(z) = p \implies F_\delta(\mathcal{C})(z) = p$$

— *Un état $q \in \mathcal{Q}$ est* quiescent *s'il n'est jamais modifié lorsque toute les cellules de son voisinage sont dans le même état q, autrement dit si*

$$\delta(q, q, \ldots, q) = q$$

Représentation condensée d'un automate cellulaire

Un automate cellulaire peut être représenté de façon condensée par la donnée de sa seule fonction de transition, si la dimension, l'ensemble d'états et le voisinage sont suffisamment clairs. Par exemple, la figure 2.1 présente la règle de transition d'un automate cellulaire de dimension $d = 1$ utilisant seulement deux états (symbolisés par des cellules vertes ou orange) et le voisinage $\mathcal{V} = \{-1, 0, 1\}$ (il s'agit du voisinage de Moore en dimension 1). Ce voisinage implique que l'état d'une cellule après application de la fonction de transition dépend uniquement de son état, de l'état de sa voisine de gauche et de celui de sa voisine de droite.

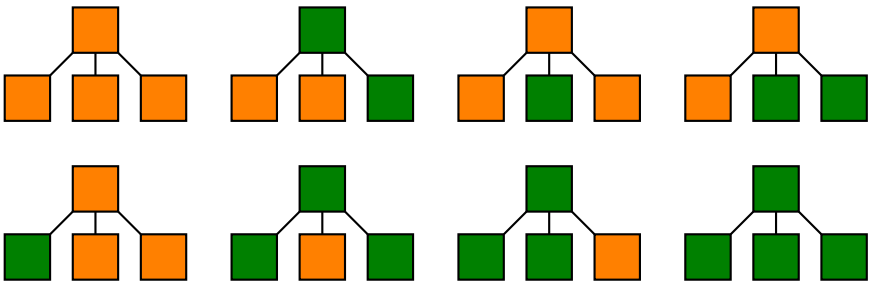


FIGURE 2.1 – Représentation condensée de la fonction de transition locale d'un automate cellulaire.

La représentation condensée se lit ainsi : les trois cases du bas représentent les états d'une cellule, de sa voisine de droite et de sa voisine de gauche ; la case du haut représente l'état de la cellule après application de la fonction de transition.

Automates cellulaires périodiques

Nous allons particulièrement nous intéresser dans cette thèse aux automates cellulaires dont la configuration initiale est périodique. Il est intéressant de noter que si la configuration initiale d'un automate cellulaire est périodique, alors toutes les configurations suivantes le sont aussi, puisqu'elles sont l'image d'une fonction périodique par une fonction appliquée uniformément.

Définition 2.8 (automate cellulaire d-périodique)**.** *On parle d'automate cellulaire* d-périodique *lorsqu'un automate cellulaire de dimension* d *s'exécute sur une configuration initiale périodique.*

Les notations suivantes introduisent les automates cellulaires d-périodiques dans deux dimensions spécifiques, qui seront les objets d'étude des deux prochains chapitres.

Notations 2.9 (automate cellulaire cyclique, automate cellulaire torique)**.** *Un* automate cellulaire cyclique *est un automate cellulaire* 1*-périodique.*

Un automate cellulaire torique *est un automate cellulaire* 2*-périodique.*

Diagramme espace-temps

Un diagramme espace-temps est une représentation de l'action d'un automate cellulaire sur une configuration, principalement utilisée pour les automates

cellulaires de dimension 1. Une configuration est représentée par une ligne de cellules, chacune possédant un état, et l'évolution de cette ligne de cellules est présentée verticalement pour chaque application de la fonction de transition. Par exemple, la figure 2.2 représente l'application de l'automate cellulaire présenté sur la figure 2.1 sur une configuration quelconque.

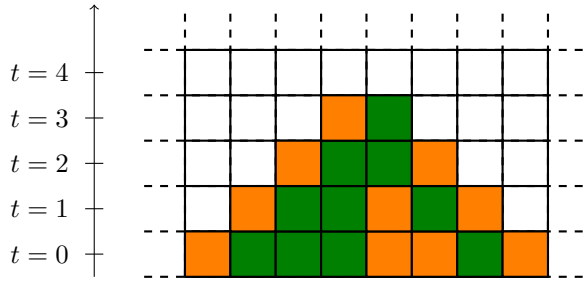


FIGURE 2.2 – Un exemple de diagramme espace-temps.

On remarque que certaines cellules sont laissées en blanc, ce qui signifie que leur état n'est pas connu. En effet, seul l'état d'un nombre fini de cellules figure sur la configuration initiale. Lorsqu'on doit appliquer la fonction de transition, les cellules au « bord » de la configuration doivent connaître l'état de leurs deux voisines pour mettre à jour le leur. Puisque ces états sont inconnus, l'état de la cellule à l'instant suivant est lui aussi inconnu. Naturellement, cette indétermination se propage à chaque application de la fonction de transition, jusqu'à ce que l'état d'aucune cellule de la configuration ne puisse être déterminé.

Dans le cas général, la connaissance d'une information finie sur la configuration ne permet de déterminer qu'une information finie dans le temps. Nous verrons dans la section suivante comment circonvenir cette propriété.

2.1.2 Calcul sur les automates cellulaires

Dans cette thèse, nous étudions les automates cellulaires en tant que *modèle de calcul*. En particulier, nous nous intéressons à la reconnaissance de langages et au calcul de fonctions. Le principe général est simple, présentons le en dimension 1 : on donne un mot fini en entrée à un automate cellulaire et on laisse cet automate appliquer itérativement sa fonction de transition pendant un certain temps, puis on lit le résultat. Les mots finis fournis en entrée à l'automate cellulaire sont des mots de l'alphabet d'entrée Σ de l'automate, qui est lui-même un sous-ensemble de son ensemble d'états $\mathcal{Q}$.

La première difficulté qui se pose est la suivante : comment fournir un mot fini en entrée à un modèle de calcul par essence infini et uniforme ? Nous allons présenter les deux solutions communément adoptées.

Entrée périodique

Une première solution est de répéter périodiquement l'entrée. La configuration est alors périodique. Pour représenter l'évolution dans le temps d'une telle configuration en dimension 1, on peut utiliser un diagramme espace-temps dans lequel on supposera que les bords droit et gauche du diagramme sont confondus (l'automate se comporte comme si sa configuration était un cycle).

La figure 2.3 présente l'évolution de la configuration initiale de la figure 2.2 si on suppose que l'entrée est présentée de façon périodique.

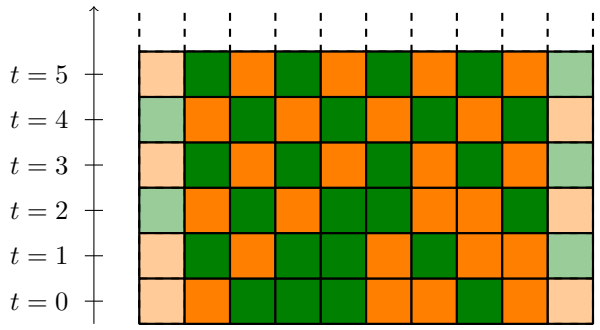


FIGURE 2.3 – Un exemple de diagramme espace-temps sur une configuration périodique (de période 8).

Entrée bornée

Une deuxième solution est d'écrire l'entrée sur un « fond » de cellules, toutes dans un état spécial $q \in \mathcal{Q}$. On ne souhaite pas que les états changent spontanément loin de l'entrée, par conséquent l'état q doit être persistant (si on souhaite que le calcul soit uniquement localisé dans la zone où le mot initial a été écrit) ou quiescent (si on autorise le calcul à s'étendre sur l'espace de travail). Évidemment, la règle de transition de l'automate doit prendre en compte ce nouvel état q.

La figure 2.4 présente l'évolution de la configuration initiale de la figure 2.2 si on suppose que l'entrée est bornée par des états persistants (figurés en gris).

Ce mode d'entrée présente la spécificité de « casser » l'uniformité du modèle de calcul : les cellules au bord de la configuration initiale peuvent maintenant jouer un rôle particulier. C'est cette spécificité qui fait qu'on utilise généralement ce type d'entrée pour reconnaître des langages sur les automates cellulaires.

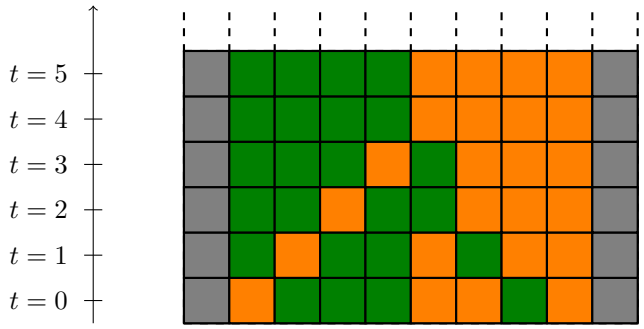


FIGURE 2.4 – Un exemple de diagramme espace-temps sur une configuration bornée par un état persistant (gris).

Remarques : La partie « déterminée » de la figure 2.2 se retrouve sur les figures 2.3 et 2.4. En effet, cette partie déterminée ne dépend pas de la méthode

d'entrée choisie.

De plus, nous remarquons que la suite des configurations successives est ultimement périodique dans les deux cas (de période 2 pour la figure 2.3, et de période 1 pour la figure 2.4). Cette propriété découle du fait que l'ensemble des configurations atteignables est fini dans les deux cas (les configurations doivent être périodiques, ou à support fini). Toute suite découlant de l'application de la fonction F_δ aux éléments d'un ensemble fini est nécessairement ultimement périodique, indépendamment de la fonction F_δ elle-même.

Reconnaissance de langage

Nous allons maintenant formaliser les notions de configuration bornée générée par un mot et de langage reconnaissable par un automate cellulaire dont l'entrée est bornée par des symboles persistants.

Définition 2.10 (configuration bornée générée par un mot)**.** *Soit $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$ un automate cellulaire de dimension $d = 1$. La configuration bornée générée par un mot $u = u_1 u_2 \dots u_n \in \Sigma^*$ est la configuration $\mathcal{C}_u$ constituée du mot u encadré par une infinité de cellules dans un états persistant ou quiescent $q \in \mathcal{Q}$. En d'autres termes, $\mathcal{C}_u(i) = u_i$ si $1 \leq i \leq n$, $\mathcal{C}_u(i) = q$ sinon.*

Il existe plusieurs formulations équivalentes d'un langage reconnaissable, voici celle que nous allons utiliser dans cette thèse.

Définition 2.11 (langage reconnaissable)**.** *Un langage $\mathcal{L} \subset \Sigma^*$ est* reconnaissable avec entrée bornée *si il existe un automate cellulaire $\mathcal{A} = (d, \mathcal{Q}, \mathcal{V}, \delta)$ et deux états distincts $acc, rej \in \mathcal{Q}$ tels que $d = 1$, $\Sigma \subset \mathcal{Q}$ et il existe une fonction $T : \mathbb{N} \to \mathbb{N}$ telle que :*

$$\begin{aligned} \forall n \in \mathbb{N},\ \forall u \in \Sigma^n,\ \forall t \geq T(n) \\ F_\delta^t(\mathcal{C}_u)(1) = acc \text{ si } u \in \mathcal{L} \\ F_\delta^t(\mathcal{C}_u)(1) = rej \text{ si } u \notin \mathcal{L} \end{aligned}$$

Cette définition s'interprète ainsi : on considère la première cellule de $\mathcal{C}_u$ ne contenant pas un symbole quiescent (elle est d'indice 1 d'après notre définition précédente). Après un certain nombre $T(n)$ d'applications de la fonction globale de transition à $\mathcal{C}_u$, cette cellule contient l'état acc (pour « acceptation ») si $u \in \mathcal{L}$, et l'état rej (pour « rejet ») sinon. L'état de la cellule ne changera plus après ce temps $T(n)$, où n est la longueur de u.

On peut également définir des notions de reconnaissance pour les automates cellulaires dont l'entrée est périodique. Toutefois, ces notions sont plus délicates et ne seront abordées qu'au chapitre suivant.

Afin d'illustrer cette définition, Nous allons présenter un automate cellulaire qui reconnaît le langage $\mathcal{L} = 0^*10^*$ des mots qui contiennent exactement une occurrence du symbole "1". Cet automate possède 6 états, dont un état persistant noté "#". Une représentation condensée de sa fonction de transition est présentée sur la figure 2.5. Le symbole "?" désigne n'importe quel état excepté l'état persistant.

Le diagramme espace-temps de cet automate appliqué à la configuration initiale générée par le mot $u = 010010$ est présenté sur la figure 2.6.

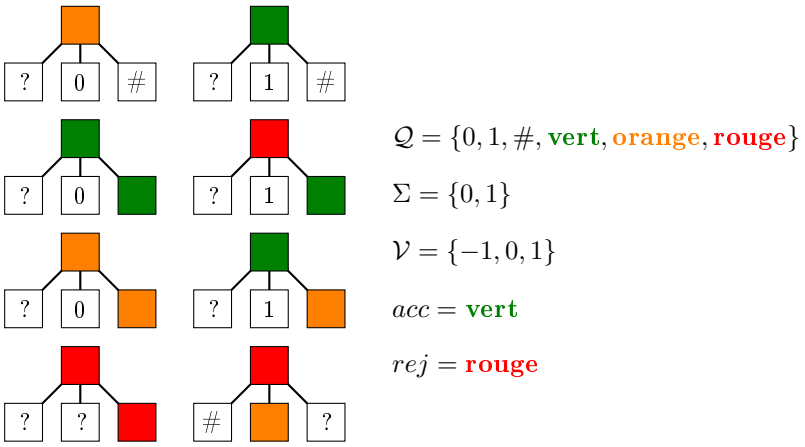


FIGURE 2.5 – Résumé de l'automate.

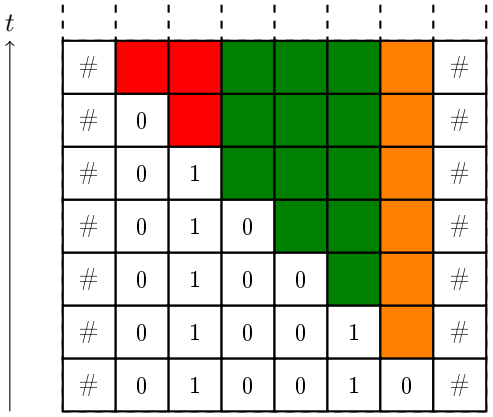


FIGURE 2.6 – Application de l'automate sur une configuration exemple.

Complexité

Une des notions les plus importantes lorsqu'on souhaite discuter des capacités de calcul d'un modèle est la notion de *complexité en temps*. Il est possible de définir cette notion sur les automates cellulaires à entrée bornée.

Définition 2.12 (complexité d'un automate cellulaire)**.** *La* complexité *d'un automate cellulaire $\mathcal{A}$ reconnaissant un langage $\mathcal{L}$ est la plus petite fonction $T : \mathbb{N} \to \mathbb{N}$ vérifiant les conditions de la définition 2.11.*

Définition 2.13 (complexité d'un langage)**.** *La* complexité *d'un langage $\mathcal{L}$ est le minimum asymptotique des complexités des automates cellulaires reconnaissant le langage $\mathcal{L}$.*

De la même manière que sur d'autres modèles de calcul (par exemple, les machines de Turing), les classes de complexité en temps des langages sur les automates cellulaires constituent un sujet de recherche intéressant et abondamment étudié (voir par exemple [50]).

Il est également possible de définir une notion de complexité en temps des automates cellulaires dont l'entrée est périodique. Toutefois, à l'instar de la notion de reconnaissance de langage sur ces mêmes automates, cette notion est délicate et sera traitée dans le chapitre suivant.

2.1.3 Signaux

Lorsqu'on souhaite construire des algorithmes élaborés sur les automates cellulaires, le concept de signal émerge naturellement. Un signal est un état, ou un sous-ensemble d'états qui se déplace sur la configuration d'un automate cellulaire et qui a pour rôle principal de transmettre une information d'un point à l'autre de la configuration. Nous nous contenterons dans cette thèse d'une description informelle du concept de signal, ainsi que d'un exemple présenté sur les figures 2.7 et 2.8, celui d'un signal effectuant des allers et retours dans une configuration bornée par des symboles persistants.

Des études approfondies du concept de signaux sur les automates cellulaires ont déjà été menées, notamment dans [44] ou [34]. Nous nous contenterons ici de signaler que les signaux peuvent avoir n'importe quelle pente rationnelle dans les limites autorisées par le voisinage de l'automate.

$$\mathcal{Q} = \{\leftarrow, \rightarrow, \#, \varnothing\}$$

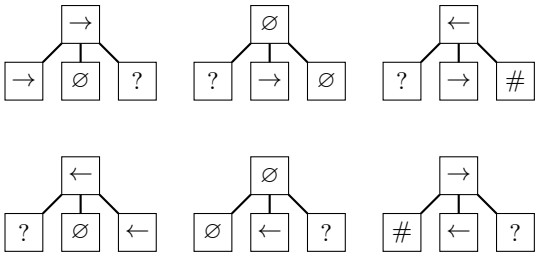


FIGURE 2.7 – Définition d'un signal élémentaire. L'état "#" est persistant, tandis que l'état "∅" est quiescent (et sera représenté par une cellule vide sur le diagramme espace-temps).

2.1.4 Constructibilité en temps

Par la suite, nous aurons parfois besoin d'une « horloge », nous utiliserons à cet effet la notion de fonction constructible en temps.

Définition 2.14 (constructibilité en temps)**.** *Une fonction croissante* $f : \mathbb{N} \to \mathbb{N}$ *est* constructible en temps *s'il existe un automate cellulaire à entrée bornée* $\mathcal{A}$*, un mot fini* $u \in \Sigma^*$ *et un état particulier* $q \in \mathcal{Q}$ *tels que*

$\forall n \in \mathbb{N}\ F_\delta^n(\mathcal{C}_u)(1) = q \iff \exists i \in \mathbb{N}\ f(i) = n$

En d'autres termes, il existe un automate qui « marque » la cellule initiale pour exactement chaque instant $f(i)$*.*

La constructibilité en temps de fonctions par des automates cellulaires est également un sujet connu et étudié, notamment dans les articles [34] ou [22]. La plupart des algorithmes qui construisent des fonctions en temps utilisent

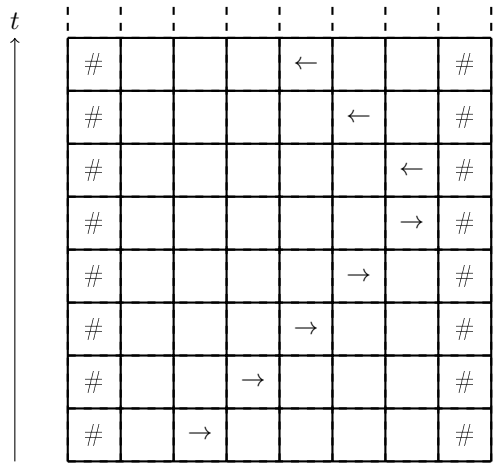


FIGURE 2.8 – Mise en œuvre du signal élémentaire.

des signaux de vitesse et de comportement variables. Par exemple, la figure 2.9 représente le diagramme espace-temps d'un automate cellulaire qui construit la fonction $f : n \mapsto n^2$. La fonction de transition complète de l'automate n'est pas explicitée, le diagramme espace-temps suffit à donner l'intuition. La construction utilise l'identité $(n+1)^2 = n^2 + 2n + 1$.

2.1.5 Synchronisation d'une configuration

Lors de la conception d'algorithmes sur les automates cellulaires, il arrive qu'on désire que toutes les cellules actives d'une configuration bornée effectuent la même opération en même temps. Si cette opération doit avoir lieu à l'initialisation de l'algorithme, aucun problème ne se pose (puisque le démarrage du calcul est en lui-même un événement synchronisant). Cependant, si cette opération doit avoir lieu à un autre instant —par exemple, lorsqu'une cellule particulière reçoit un signal— alors un problème se pose : comment déclencher une opération simultanée sur une nombre arbitrairement grand de cellules, alors que le voisinage de l'automate est fini ?

Ce problème de synchronisation d'un ensemble de cellules, également appelé « problème de la ligne de fusiliers » (ou « firing squad synchronization problem ») en dimension 1, se formalise de la manière suivante :

> *Considérons la configuration d'un automate cellulaire bornée par des symboles persistants, et composée d'états quiescents (état « soldat ») à l'exception de la cellule du bord gauche de la configuration (dans un état « général »). Comment spécifier l'ensemble des états de l'automate et sa fonction de transition afin qu'à un certain instant toutes les cellules entrent en même temps dans un état particulier jamais apparu auparavant (l'état « feu »).*

Le problème de la ligne de fusiliers a été largement étudié (voir par exemple [32]), et plusieurs solutions en temps minimal ont été élaborées, notamment dans [31]. Nous présenterons dans la prochaine section une utilisation d'un algorithme

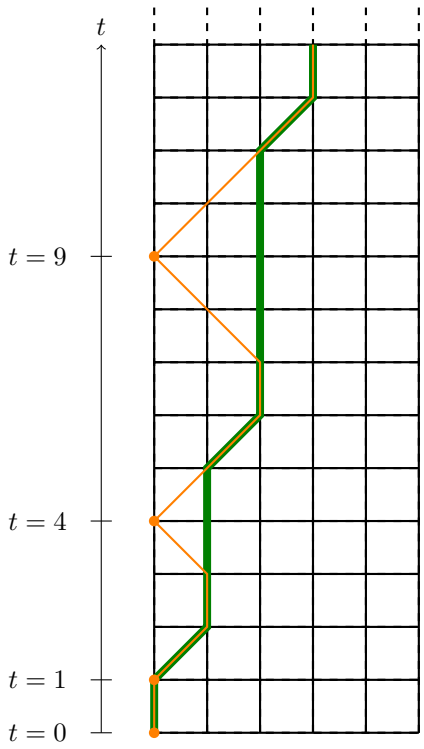


FIGURE 2.9 – Construction en temps de la fonction n^2 en utilisant deux signaux. Pour des raisons de lisibilité, les cellules sont représentées comme des points de la grille, et les signaux comme des segments entre ces points.

simple qui résout le problème de la ligne de fusiliers en utilisant des signaux de différentes vitesses.

2.2 Exemple préliminaire : le problème du compactage

Nous allons maintenant présenter un algorithme sur les automates cellulaires de dimension 2 à entrée bornée par des symboles persistants. Cet algorithme résout le problème du compactage en temps linéaire et a initialement été présenté dans [1].

Le problème du compactage a été énoncé par S. R. Kosaraju dans [26] de la façon suivante :

> *Étant donnée une image carrée $n \times n$ composée de cellules noires ou blanches (on a donc $\Sigma = \{noir, blanc\}$) et encadrée par des symboles persistants, comment définir la règle de transition d'un automate cellulaire afin de la transformer en une image de même dimension où le nombre de cellules de chaque couleur est conservé, mais où les cellules noires sont entassées en lignes pleines en haut de l'image (sauf éventuellement la dernière ligne, alignée à gauche).*

Des versions légèrement différentes du problème du compactage ont également été étudiées dans le contexte des circuits VLSI (*Very Large Scale Integration*), notamment dans [47] ou [28].

La résolution de ce problème a de nombreuses applications, par exemple en traitement d'image, ou pour la reconnaissance du langage de la « majorité », composé des images possédant plus de pixels noirs que de pixels blancs. On peut également l'interpréter du point de vue du regroupement de l'information : si on considère que les cellules noires portent une information particulière en plus de leur couleur et que les cellules blanches servent de « fond » à un calcul, il peut être désirable de regrouper tous les états des cellules noires au même endroit afin d'accélérer un éventuel traitement.

Une fonction de transition simple résolvant ce problème est présentée par Kosaraju dans [26]. Nous présentons cette fonction sur la figure 2.10 ; elle peut être résumée ainsi : À l'initialisation de l'algorithme, on écrit une flèche (représentée par un état) dans chaque cellule de la configuration. Les cellules des lignes paires auront une flèche orientée vers la droite, et celles des lignes impaires seront orientées vers la gauche. Ensuite, les couleurs se déplacent comme suit : les états noirs se déplacent vers le haut si la cellule au-dessus d'eux est blanche. Sinon, ils se déplacent dans le sens indiqué par la flèche si la cellule ciblée est blanche. Lorsqu'une cellule blanche peut recevoir un état noir depuis la cellule du bas ou depuis une cellule à côté d'elle, la cellule du bas a la priorité.

Kosaraju émettait la conjecture erronée que cet algorithme résolvait le problème du compactage en temps linéaire en le périmètre de l'image. Cependant, il est aisé de se convaincre de l'inexactitude de cette conjecture en étudiant l'exécution de l'algorithme sur une image dont la moitié droite est noire et la moitié gauche est blanche. On se propose donc dans cette section de présenter un autre algorithme, qui résout le problème du compactage en temps linéaire.

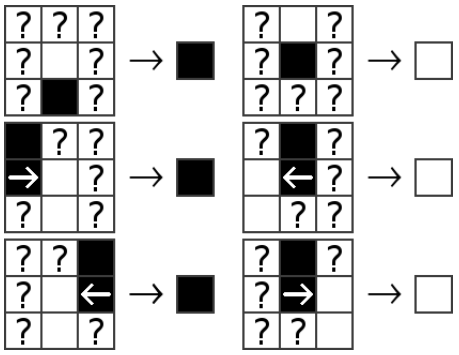


FIGURE 2.10 – Les règles de compactage de Kosaraju (utilisant le voisinage de Moore en dimension 2.

Plus précisément, nous allons améliorer l'algorithme précédent en utilisant une approche de type « diviser pour régner ». Nous allons établir deux points essentiels :

— De manière générale, il est possible d'implémenter des algorithmes de type « diviser pour régner » sur les automates cellulaires. Nous verrons comment diviser l'espace initial et comment réaliser la réunification propre des sous-parties, le tout en temps linéaire.
— Étant donnée la configuration d'un automate cellulaire sous la forme d'une image carrée $n \times n$ découpée en 4 sous-images $n/2 \times n/2$, chacune de ces sous-images étant compactée, il est possible de résoudre le problème du compactage en temps $O(n)$ sur cette image.

Si nous réussissons à prouver ces deux points, nous aurons prouvé que notre algorithme résout le problème du compactage en temps linéaire. En effet, le facteur logarithmique qui apparaît souvent dans les analyses d'algorithmes séquentiels « diviser pour régner » sera dans notre cas absorbé par le traitement parallèle des 4 sous-images de la configuration.

2.2.1 Diviser pour régner sur les automates cellulaires

La méthode générale de diviser pour régner est une manière efficace de résoudre de nombreux problèmes sur les modèles de calcul séquentiels. De plus, elle apparaît encore plus efficace sur les automates cellulaires, car les différents sous-problèmes peuvent être traités en même temps, grâce à la nature massivement parallèle du modèle. La principale difficulté rencontrée lors de la conception d'algorithmes de type diviser pour régner est le besoin d'un processus global pour diviser le problème en sous-problèmes puis pour fusionner les sous-problèmes proprement une fois qu'ils ont été résolu. Un tel processus global est délicat à mettre en place sur le modèle de calcul local qu'est celui des automates cellulaires.

Il est intéressant de noter que le problème de division et de fusion de la zone de calcul n'a besoin d'être résolu que pour les automates cellulaires de dimension 1. En effet, si un tel algorithme de découpage et de fusion est connu, il suffit de l'appliquer en parallèle sur toutes les dimensions de l'automate pour obtenir un algorithme « diviser pour régner » en dimension quelconque.

Dans les sections suivantes, nous verrons comment effectuer ces processus de division et de fusion en adaptant un algorithme naïf qui résout le problème de la ligne de fusiliers. Nous allons supposer dans un souci de simplicité que les dimensions de l'espace de calcul (celui qui est borné par des symboles persistants) sont des puissances de 2. Nous appellerons « frontière » l'espace entre deux cellules adjacentes. Pour des raisons d'élégance, les diagrammes espace-temps seront conçus comme si les frontières de la configuration pouvaient envoyer des signaux. Les informations portées par ces signaux seront effectivement transmises par les cellules partageant la frontière.

Division de la zone de calcul

De nombreux algorithmes existants divisent une configuration unidimensionnelle. Nous allons en présenter un qui est communément utilisé dans un algorithme naïf de ligne de fusiliers, qui a été introduit pour la première fois dans [36] (voir la figure 2.11 pour une meilleure compréhension) :

1. Depuis la cellule la plus à gauche, envoyer deux signaux de vitesses respectives 1 et $1/3$. Le signal de vitesse 1 rebondit contre les symboles persistants délimitant la configuration.
2. Lorsque les deux signaux s'interceptent (au milieu de la configuration par construction, voir la figure 2.11), envoyer un signal vertical (de vitesse 0) qui agira comme une frontière pour les futurs signaux. Envoyer ensuite une nouvelle paire de signaux de pente 1 et $1/3$ de chaque côté de cette nouvelle frontière. On marquera aussi la frontière d'un état 'L' si le signal de pente 1 venait de la gauche, ou 'R' s'il venait de la droite.

- Répéter l'étape 2 jusqu'à ce que chaque cellule soit isolée.

Note : Lorsqu'on utilise cette méthode pour résoudre le problème de la ligne de fusiliers, il suffit qu'une cellule entre dans l'état « feu » lorsqu'elle est isolée. Par construction, les cellules deviennent toutes isolées en même temps, le problème est ainsi résolu.

Fusion des zones de calcul

Nous utilisons pour la fusion le mot sur l'alphabet $\{L, R\}$ produit par l'algorithme de division. On assigne à chaque frontière le symbole qui lui correspond, puis on marque chaque frontière de position impaire (voir la figure 2.12). On répète ensuite les étapes suivantes :

— Attendre la fin du traitement de niveau inférieur dans chacune des sous-parties.

— Supprimer les frontières marquées, puis envoyer un signal de pente 1 dans la direction indiquée par le symbole de la frontière, et un signal de pente $1/3$ dans la direction opposée. Ces signaux rebondissent contre les frontières. Lorsque 4 signaux s'intersectent sur une frontière, marquer cette frontière et supprimer les 4 signaux.

Ce mécanisme nous assure que les frontières sont bien supprimées dans l'ordre inverse de leur création, ce qui est nécessaire aux algorithmes « diviser pour régner » utilisant ces frontières (voir figure 2.12).

Complexité temporelle de l'algorithme

Si on suppose que chaque sous-traitement (par exemple, le compactage) est effectué en un temps linéaire en la largeur de sa sous-image, et si notre image initiale est de taille $n \times n$, alors il existe une constante C telle que la complexité globale de l'algorithme est :

$$C \times n + C \times \frac{n}{2} + C \times \frac{n}{4} + \cdots + C = O(n)$$

2.2.2 L'algorithme de compactage

Avant de lancer l'algorithme proprement dit, il est nécessaire d'attribuer une direction à chaque cellule de l'image. Chaque cellule d'un rang impair contiendra une flèche orientée vers la droite, et chaque cellule d'un rang pair contiendra une flèche orientée vers la gauche. Ces flèches peuvent être écrites en un temps linéaire en la longueur du côté de l'image, et ne seront plus modifiées une fois écrites.

Au cours de la description de l'algorithme, on dira que les états noirs sont « tassés » dans une direction particulière lorsqu'on exécute un algorithme de compactage uni-dimensionnel dans cette direction. Les règles de transition permettant d'effectuer un tassage sont simples : si un état noir a pour voisin un état blanc dans la direction du tassage, les deux états échangent leur position.

Nous allons maintenant illustrer comment fusionner 2×2 images compactées de dimension $k \times k$ en une seule image carrée compactée de taille $2k \times 2k$. Une configuration initiale d'un tel problème est présentée sur la figure 2.13.

Étape 1

La première chose à faire et de tasser les lignes complètement remplies des carrés initiaux en haut de l'image, et les lignes partiellement remplies en bas de l'image. On peut différencier ces lignes, par exemple en envoyant un signal qui parcourt les lignes des carrés initiaux et vérifie si elles contiennent ou non des états blancs. Deux couches d'états sont utilisées pour ce calcul : une couche traitera les lignes pleines, et une autre traitera les lignes partiellement remplies. Lorsque ce tassage est terminé, l'image est constituée d'au plus 3 composantes connexes (voir figure 2.14). De plus, il est clair que les états noirs dans la couche traitant les lignes partiellement remplies sont disposés dans au plus deux lignes.

Étape 2

On veut maintenant équilibrer les demi-lignes pleines en haut de l'image, de telle sorte qu'il n'y ait plus qu'au maximum une ligne incomplète en haut de l'image. Pour cela, il suffit de tasser chaque ligne de la couche contenant les lignes pleines selon la direction inscrite dans ses cellules (voir figure 2.15), puis de tasser les cellules de cette même couche vers le haut (voir figure 2.16).

Finalement, on fusionne les deux couches de calcul, puis on tasse les états vers le haut, pour obtenir une image telle que présentée sur la figure 2.17. Notons que cette image contient au maximum trois lignes incomplètes, et que le nombre de cellules dans ces lignes incomplètes est croissant de bas en haut.

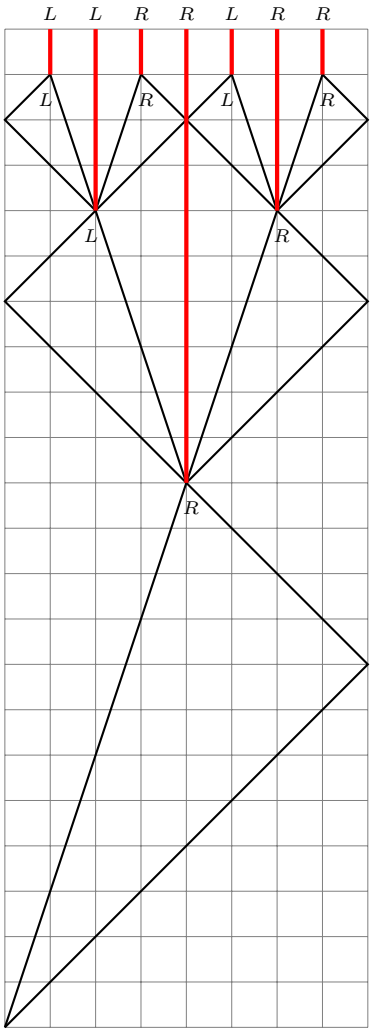


FIGURE 2.11 – L'algorithme de division de la ligne de fusiliers.

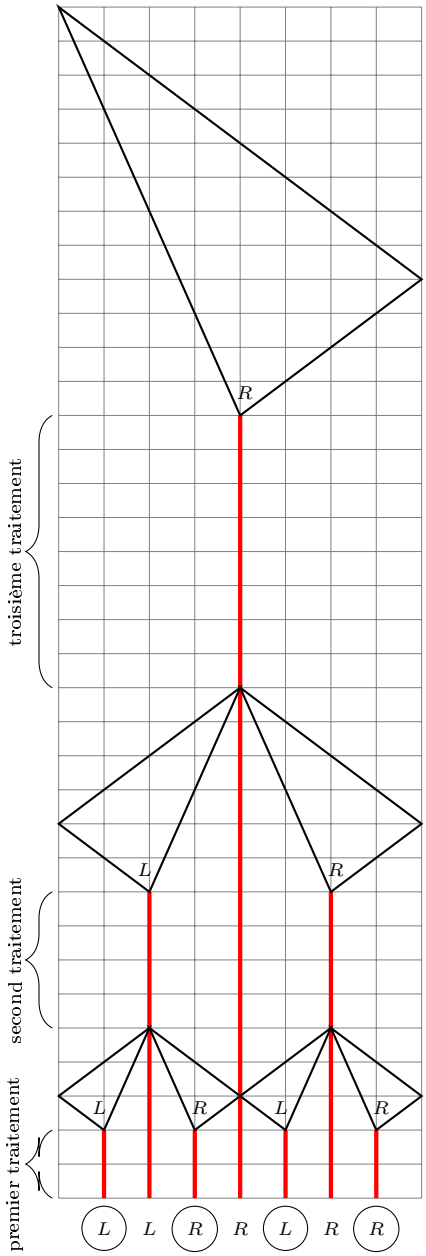


FIGURE 2.12 – L'algorithme de fusion des sous-images.

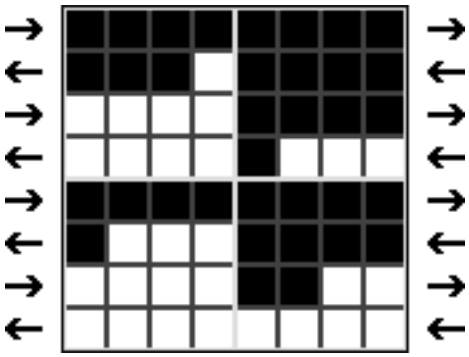


FIGURE 2.13 – Une configuration composée de 4 images carrées individuellement compactées.

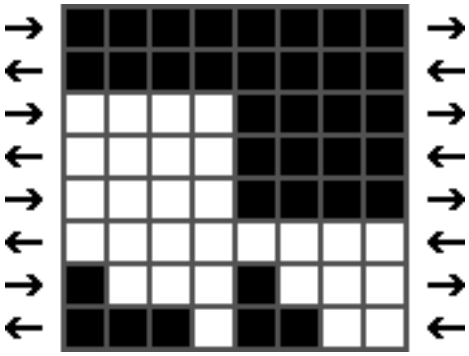


FIGURE 2.14 – Après le premier tassage.

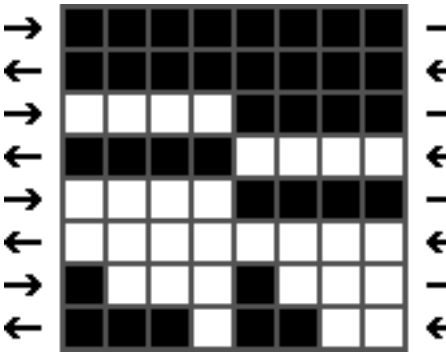


FIGURE 2.15 – Tassage selon la ligne

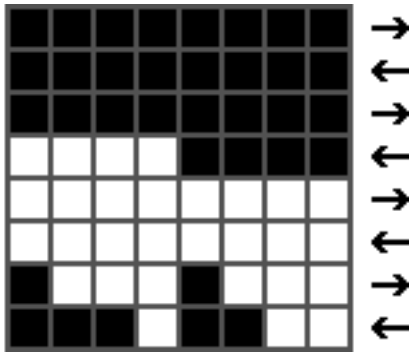


FIGURE 2.16 – ... puis vers le haut.

Étape 3

L'algorithme va maintenant répéter les premières pas de l'étape 2 — c'est-à-dire tasser les cellules selon la direction de leur ligne, puis les tasser vers le haut — jusqu'à obtenir une image contenant au plus une ligne incomplète. Nous allons prouver que deux répétitions de ce processus suffisent à obtenir une telle image.

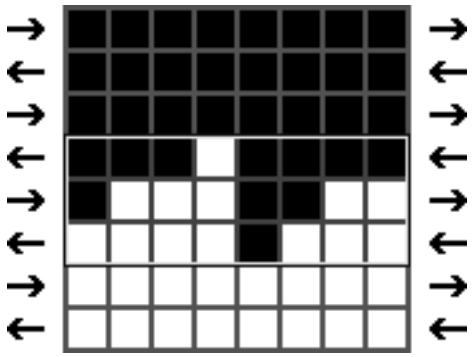


FIGURE 2.17 – L'image à la fin de l'étape 2.

En effet, par construction les trois lignes incomplètes sont consécutives. Nous allons montrer que dans cette situation, chaque exécution des deux étapes indiquées ci-dessus ont pour effet soit de vider le rang du bas, soit de remplir le rang du haut :

Soit n la longueur de nos lignes, et n_1, n_2, n_3 le nombre d'états noirs dans la ligne du haut (ligne 1), du milieu (ligne 2) et du bas (ligne 3) respectivement. Nous avons noté précédemment que $n_1 \geq n_2 \geq n_3$.

Supposons maintenant que la ligne 1 n'ait pas été remplie par la première série de tassage selon les lignes puis vers le haut. Puisque les rangs 1 et 2 ont été tassés dans des directions différentes, cela signifie que $n_1 + n_2 < n$, nous avons par conséquent $n_2 < n/2$. Puisque $n_2 \geq n_3$ cela signifie que nous avons également $n_3 < n/2$, et finalement $n_2 + n_3 < n$. Cette inégalité nous assure qu'il y a assez de place dans la ligne 2 pour stocker les états noirs de toutes les cellules des lignes 2 et 3, la ligne 3 est par conséquent vide après la première série de tassages.

Un raisonnement identique nous permet de conclure que la seconde série de tassages ne laisse qu'une seule ligne incomplète, ce qui conclut notre preuve. Notons que dans notre exemple une seule série de tassages est suffisante pour à la fois remplir la ligne 1 et vider la ligne 3 (voir les figures 2.18 et 2.19).

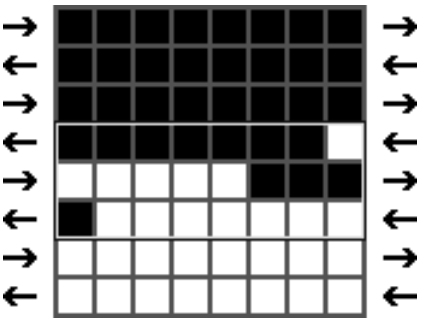


FIGURE 2.18 – Tassage de la ligne

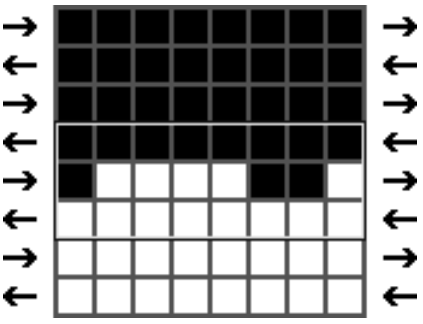


FIGURE 2.19 – ...puis vers le haut.

Il suffit maintenant de tasser la dernière ligne incomplète vers la gauche pour obtenir une image compactée (voir figure 2.20).

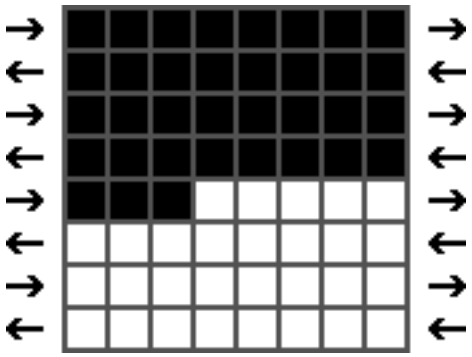


FIGURE 2.20 – L'image terminale.

Complexité temporelle

Chacune des opérations détaillées ci-dessus peut être effectuée en un temps linéaire en la longueur du côté du carré contenant le sous-problème en cours de résolution. Par conséquent, il est possible de fusionner 4 images compactées en une seule en un temps linéaire en la longueur du côté de l'image résultante.

En tenant compte de notre remarque précédente sur la complexité générale des algorithmes « diviser pour régner » sur les automates cellulaires, nous avons bien prouvé que l'algorithme que nous avons présenté résout le problème du compactage en temps linéaire.

Remarques finales

Dans l'article à l'origine du problème du compactage [26] l'auteur présente un deuxième algorithme pour résoudre le problème du compactage, et prouve qu'il s'exécute en temps linéaire. Cet algorithme procède par comptage des états noirs sur chaque ligne, puis par sommation et par réécriture des états noir en bon ordre à partir de la cellule en haut à gauche de la configuration.

À la lumière de cette remarque, on pourrait penser que l'algorithme que nous venons de présenter n'est qu'un prétexte pour présenter les différents mécanismes qu'il est possible d'utiliser sur les automates cellulaires. Toutefois il n'en est rien, notre algorithme présentant entre autres avantages celui de conserver les états noirs à tout instant. Cette propriété s'avère notamment utile si on considère le problème du compactage comme une manière de regrouper l'information en vue d'un traitement futur, et que les états noirs sont une manière simplifiée de représenter les états portant une information utile. Dans ce contexte, le comptage des informations induirait une croissance exponentielle du nombre d'états nécessaire, alors que note algorithme se contente de multiplier leur nombre par un facteur constant.

Chapitre 3

Configurations périodiques de dimension 1 : période minimale et classes de complexité

Dans ce chapitre, nous nous intéressons aux algorithmes et à la complexité des automates cellulaires cycliques (de dimension 1), c'est à dire agissant sur des configurations périodiques. L'entrée d'un tel automate est un mot circulaire, autrement dit un mot défini à permutation circulaire près. Nous verrons qu'un langage reconnu par un automate cellulaire cyclique est un langage cyclique [11], c'est-à-dire, un langage clos par shift, puissance et racine.

La première question qui se pose est de trouver la « période minimale » d'une configuration périodique. Notre problème de référence, appelé PERIODE-MINIMALE est le suivant : étant donnée une configuration périodique, calculer une période minimale ; de façon équivalente, étant donné un mot circulaire w, calculer sa racine canonique (ou mot primitif) u, c'est-à-dire, le mot minimal u tel que w égale u^p, pour un entier p. Nous construisons un algorithme sur un automate cellulaire cyclique qui calcule la période minimale en temps polynomial.

Ce résultat est fondamental pour comparer la complexité des automates cycliques avec la complexité des automates cellulaires standard, c'est-à-dire, dont le mot d'entrée est entouré de symboles spéciaux. Nous prouvons que les classes de complexité coïncident dans les deux modèles, au temps polynomial près. Plus précisément, nous démontrons les équivalences suivantes, pour tout langage cyclique L :

- L est reconnu sur un automate cellulaire cyclique si et seulement si L est reconnu en espace linéaire ($L \in LINSPACE$) ;
- L est reconnu sur un automate cellulaire cyclique en temps polynomial si et seulement si L est reconnu sur un automate cellulaire standard en espace linéaire et temps polynomial.

Par ailleurs, nous étendons de façon naturelle ces résultats de complexité aux fonctions et en déduisons que le problème très étudié de la densité, « Density

Classification Problem » [27], peut être résolu (en temps polynomial) si on autorise un ensemble d'états de calcul plus grand que l'alphabet d'entrée $\{0, 1\}$.

3.1 Généralités sur les configurations périodiques

Dans cette section, nous étudions les spécificités des automates cellulaires travaillant sur des configurations périodiques en dimension 1, par opposition aux automates dits « classiques », dont la configuration initiale est composée d'un mot fini borné par des symboles persistants. Nous insisterons dans un premier temps sur les différences et les points communs entre ces deux modes d'entrée, puis nous étudierons les restrictions imposées par le modèle périodique sur les mots et les langages qu'un automate cellulaire peut reconnaître.

3.1.1 Entrée bornée, entrée périodique

Nous nous intéressons dans les deux cas à des automates cellulaires qui traitent des configurations *à support fini*. Dans le cas d'un automate à entrée bornée, la définition de la configuration générée par un mot donné est triviale. Nous donnons ici la définition d'une configuration périodique générée par un mot fini donné :

Définition 3.1 (configuration périodique générée par un mot)**.** *La* configuration générée par le mot fini $u = u_0u_1 \dots u_{|u|-1}$ *est la configuration périodique* $\mathcal{C}_u$ *issue de la répétition du mot* u *à l'infini. Autrement dit,* $\forall i \in \mathbb{Z} \quad \mathcal{C}_u(i) = u_{i \mod |u|}$.

Un objectif clé est de déterminer la période minimale d'une configuration périodique.

Définition 3.2 (période minimale d'une configuration)**.** *La* période minimale *d'une configuration périodique* $\mathcal{C} \in \Sigma^{\mathbb{Z}}$ *est le mot* $u \in \Sigma^*$ *de longueur minimale tel que* $\mathcal{C} = \mathcal{C}_u$.

Nous devons remarquer que certaines configurations générées par des mots différents vont évoluer de la même manière lorsqu'elles seront fournies en entrée à un automate cellulaire. Des exemples de telles configurations sont présentés sur la figure 3.1 : ces configurations sont identiques, à translation près.

La règle de transition d'un automate cellulaire est locale et uniforme, et ne prend pas en compte la position absolue de la cellule à laquelle on l'applique. Par conséquent, les diagrammes espace-temps des configurations présentées dans la figure 3.1 seront identiques à translation près, et ce quelle que soit la règle de transition (et donc l'automate cellulaire) qui s'y applique.

3.1.2 Langages cycliques

Nous allons maintenant formaliser l'intuition que nous avons eu précédemment, à savoir que certains mots doivent se comporter de la même manière que d'autres. Nous introduisons tout d'abord quelques définitions formelles :

Définition 3.3 (puissance d'un mot, racine d'un mot)**.** *Soient deux mots* $u, v \in \Sigma^+$, v *est une* puissance *de* u *(et* u *est une racine de* v*) si* $\exists k \in \mathbb{N}^* \; v = u^k$.

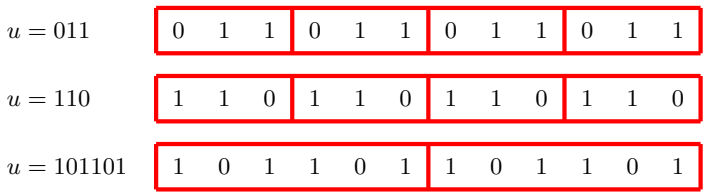


FIGURE 3.1 – Ces configurations générées par des mots différents vont toutes évoluer de la même manière.

Définition 3.4 (mot primitif). *Un mot $u \in \Sigma^+$ est* primitif *s'il n'est la puissance d'aucun autre mot de Σ^+, autrement dit si $\forall v \in \Sigma^+ \forall k > 1\, u \neq v^k$.*

Définition 3.5 (fonction shift). *La* fonction shift *notée $\sigma : \Sigma^* \to \Sigma^*$ est définie sur l'ensemble des mots par $\forall u = u_1 u_2 \dots u_n \in \Sigma^* \quad \sigma(u) = u_2 \dots u_n u_1$.*

Nous nous intéressons aux automates cellulaires périodiques du point de vue de la *reconnaissance de langages*. Notre intuition précédente se formalise ainsi : si un automate reconnaît un mot (nous verrons ce que signifie exactement cette notion plus tard), alors il doit reconnaître les puissances de ce mot, les racines de ce mot et les shifts de ce mot, puisque les configurations qui en résultent sont équivalentes.

Par conséquent, tout langage « reconnu » par un automate cellulaire cyclique doit être clos par shift, translation et racine. De tels langages sont appelés *cycliques*, et ont déjà été définis et étudiés dans le cadre des langages rationnels, notamment dans un article de Carton [11]. En voici une définition formelle :

Définition 3.6 (langage cyclique). *Un langage $L \subset \Sigma^*$ est dit* cyclique *si $\forall w \in \Sigma^*, \forall k \geq 1$:*

- *$w \in L \iff \sigma^k(w) \in L$ (L est stable par shift),*
- *$w \in L \iff w^k \in L$ (L est stable par puissance et racine).*

Tout langage cyclique peut également être défini comme la clôture d'un ensemble (potentiellement infini) de mots primitifs par les opérations de puissance et de shift. Ces langages cycliques sont donc les seuls qu'il est possible de « reconnaître » sur un automate cellulaire cyclique.

3.2 Reconnaissance cyclique et algorithme de partitionnement

Dans cette section, nous allons définir formellement la notion de reconnaissance d'un langage cyclique. Nous allons étudier les liens qu'elle entretient avec la reconnaissance classique effectuée à l'aide d'un automate cellulaire à entrée bornée. En particulier, nous énoncerons un résultat d'équivalence que nous prouverons à l'aide d'un algorithme qui va partitionner une configuration périodique en mots primitifs.

3.2.1 Reconnaissance de mots sur les automates cellulaires cycliques

Sur un automate cellulaire standard où l'entrée est bornée par des symboles persistants, il est possible d'identifier une cellule particulière (par exemple, la plus à gauche), et s'en servir pour porter le résultat d'un calcul. Un mot donné en entrée à cet automate sera considéré comme *accepté* (resp. *rejeté*) quand cette cellule précise entre dans un état particulier de $\mathcal{Q}$ dit *état d'acceptation* (resp. *état de rejet*). Une étude plus complète de la reconnaissance de langages par les automates cellulaires peut être trouvée dans l'article de Terrier [50].

En revanche, dans le cadre d'un automate cellulaire cyclique, il est impossible d'identifier une cellule en particulier : une telle « marque » briserait la nature périodique de la configuration. Par conséquent, l'acceptation ou le rejet d'un mot d'un langage doit être un phénomène global. Voici les deux définitions d'acceptation que nous allons utiliser sur les automates cellulaires cycliques :

Définition 3.7 (reconnaissance faible, reconnaissance forte)**.** *Un langage $L \subset \Sigma^*$ est* cycle-reconnaissable *s'il existe un automate cellulaire cyclique $\mathcal{A}$ et deux sous-ensembles disjoints d'états de $\mathcal{A}$, $\mathcal{Q}_a$ (ensemble d'états acceptants) et $\mathcal{Q}_r$ (ensemble d'états de rejet), tels que :*

*Pour tout $u \in \Sigma^+$, l'automate $\mathcal{A}$ travaillant sur la configuration périodique $\mathcal{C}_u$ évolue de manière telle qu'après un certain temps t toutes les cellules entrent dans l'ensemble $\mathcal{Q}_a$ si $u \in L$ (resp. $\mathcal{Q}_r$ si $u \notin L$) et restent dans cet ensemble après t (**reconnaissance faible**).*

*Si en plus les ensembles d'états acceptants et de rejet sont réduits aux singletons $\mathcal{Q}_a = \{q_a\}$ et $\mathcal{Q}_r = \{q_r\}$, alors on parle de **reconnaissance forte**.*

Nous remarquons que si un automate cellulaire cyclique $\mathcal{A}$ reconnaît *fortement* un langage, alors il converge nécessairement vers les configurations $\mathcal{C}_{q_a}$ ou $\mathcal{C}_{q_r}$, qui sont des points fixes de la fonction de transition globale $F_{\mathcal{A}}$.

Nous déduisons immédiatement de cette définition et de nos observations précédentes le fait suivant :

Fait 3.8. *Si un langage L est cycle-reconnaissable, faiblement ou fortement, alors L est un* langage cyclique.

Pour aider à la compréhension des notions de reconnaissances faible et forte, nous présentons deux automates cellulaires sur les figures 3.2 et 3.4, ainsi que des exemples de diagrammes espace-temps sur les figures 3.3 et 3.5.

$\mathcal{Q} = \{0, 1, 2, \phi\}$ $\quad \mathcal{Q}_a = \{1, \phi\}$ $\quad \mathcal{Q}_r = \{2, 0\}$
$\mathcal{V} = \{-1, 0, 1\}$

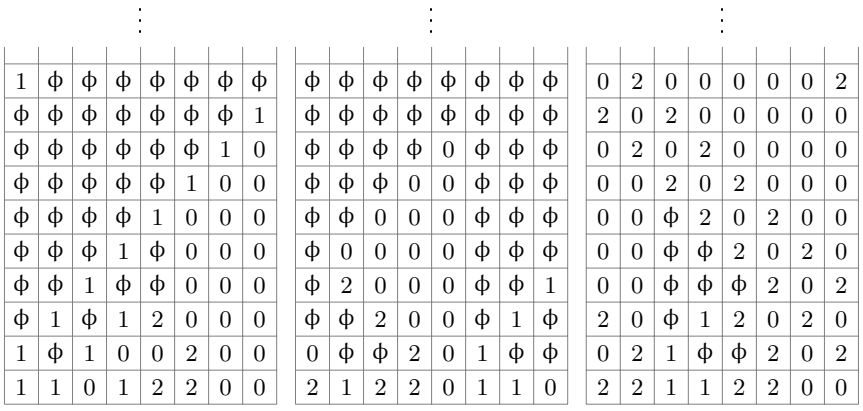
0	0	1	2	ϕ
0	0	0	2	0
1	1	1	ϕ	1
2	0	0	2	0
ϕ	ϕ	ϕ	2	ϕ

1	0	1	2	ϕ
0	ϕ	1	ϕ	ϕ
1	ϕ	1	ϕ	ϕ
2	ϕ	1	ϕ	ϕ
ϕ	ϕ	1	ϕ	ϕ

2	0	1	2	ϕ
0	0	0	0	0
1	ϕ	ϕ	ϕ	ϕ
2	2	2	2	2
ϕ	0	0	0	0

ϕ	0	1	2	ϕ
0	ϕ	ϕ	2	ϕ
1	1	1	ϕ	1
2	ϕ	ϕ	2	ϕ
ϕ	ϕ	ϕ	2	ϕ

FIGURE 3.2 – L'automate $\mathcal{A}_{faible}$ qui reconnaît faiblement le langage $L_{\#1 \geq \#2}$.



(a) $\#1 > \#2$ (b) $\#1 = \#2$ (c) $\#1 < \#2$

FIGURE 3.3 – Exemples d'application de l'automate $\mathcal{A}_{faible}$ sur différentes configurations périodiques.

Notes concernant les figures 3.2 et 3.3

L'automate présenté dans la figure 3.2 reconnaît faiblement le langage cyclique $L_{\#1 \geq \#2}$ composé des mots sur l'alphabet $\{0, 1, 2\}$ dont le nombre de 1 est supérieur ou égal au nombre de 2. Il utilise également un quatrième état noté ϕ (à interpréter comme « un 0 qui a rencontré un 1 »). Sa fonction de transition utilise le voisinage $\mathcal{V} = \{-1, 0, 1\}$, est exprimée sous forme tabulaire, et peut se résumer comme suit :

- les 1 se déplacent vers la droite en remplaçant les 0 par des ϕ jusqu'à rencontrer un 2 ;
- les 2 se déplacent vers la gauche en remplaçant les ϕ par des 0 jusqu'à rencontrer un 1 ;
- lorsqu'un 1 rencontre un 2, ils sont supprimés et remplacés par un ϕ ;
- les ϕ se propagent vers la droite en remplaçant les 0.

Les règles se lisent comme suit :

c	d
g	$\delta(g, c, d)$

L'ensemble des états d'acceptation est $\mathcal{Q}_a = \{1, \phi\}$, l'ensemble des états de rejet est $\mathcal{Q}_r = \{2, 0\}$. On remarque que chacun des diagrammes espace-temps de la figure 3.3 entre dans une boucle dans laquelle les états appartiennent tous au même sous-ensemble d'acceptation ou de rejet.

$\mathcal{Q} = \{0,1\}$ $\mathcal{Q}_a = \{1\}$ $\mathcal{Q}_r = \{0\}$
$\mathcal{V} = \{-1,0,1\}$

0	0	1
0	0	1
1	1	1

1	0	1
0	1	1
1	1	1

FIGURE 3.4 – L'automate $\mathcal{A}_{fort}$ qui reconnaît fortement le langage $L_{\exists 1}$ des mots sur $\Sigma = \{0,1\}$ qui contiennent un 1.

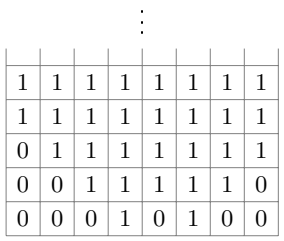


FIGURE 3.5 – Exemple d'application de l'automate $\mathcal{A}_{fort}$ sur une configuration périodique.

Par la suite, nous omettrons de préciser qu'il s'agit de reconnaissance cyclique quand il n'y aura pas d'ambiguïté ; nous parlerons simplement de reconnaissance faible ou forte. De plus, nous prouverons par la suite que ces notions sont équivalentes en dimension 1.

Notons que notre définition de reconnaissance forte est d'une certaine manière la « meilleure » définition de reconnaissance qui puisse être obtenue, à cause de la nature périodique de notre modèle. Cette même nature nous amène à faire une remarque utile sur la borne supérieure du temps de reconnaissance d'un langage.

Fait 3.9. *Si un automate cellulaire cyclique reconnaît un langage L, alors le temps t après lequel les cellules sont dans leur sous-ensemble d'états terminaux $\mathcal{Q}_a$ ou $\mathcal{Q}_r$ est borné par une fonction exponentielle en la longueur n du mot u testé.*

En effet, chaque configuration étant, comme la configuration initiale $\mathcal{C}_u$ de période n, le nombre de configurations possibles distinctes est exponentiel en n. Si l'automate évolue pendant un temps supérieur à ce nombre exponentiel de configurations, il entre nécessairement dans un cycle.

3.2.2 Calculabilité de fonctions sur automates cellulaires cycliques

Comme dans la plupart des modèles de calcul, il est naturel de généraliser les automates cellulaires cycliques pour le calcul de fonctions : la reconnaissance de langage est alors un cas particulier de fonctions. Intuitivement, nous voudrions fournir l'entrée d'une fonction sous la forme d'une entrée périodique à l'automate cellulaire, attendre un certain temps, puis lire le résultat de la fonction lorsque l'automate a atteint un point fixe. C'est le sens de la définition suivante :

Définition 3.10 (fonction cycle-calculable)**.** *Soit Σ et Γ deux alphabets disjoints. Une fonction $f : \Sigma^+ \to \Gamma^+$ est* cycle-calculable *s'il existe un automate cellulaire cyclique $\mathcal{A}$ tel que pour tout $u \in \Sigma^+$:*

— *$\exists t \in \mathbb{N}$ tel que $F^t_{\mathcal{A}}(\mathcal{C}_u) = \mathcal{C}_{f(u)}$ (l'automate calcule $f(u)$),*

— *$F_{\mathcal{A}}(\mathcal{C}_{f(u)}) = \mathcal{C}_{f(u)}$ ($\mathcal{C}_{f(u)}$ est un point fixe de $F_{\mathcal{A}}$).*

Le temps *de calcul de $\mathcal{A}$ sur u est le plus petit entier t tel que $\mathcal{C}_{f(u)} = F^t_{\mathcal{A}}(\mathcal{C}_u)$.*

Nous pouvons immédiatement déduire quelques propriétés des fonctions *cycle-calculables* à partir de la définition précédente. Nous nous servons de ces propriétés pour établir une définition similaire à celle des langages cycliques, appliquée aux fonctions :

Définition 3.11 (fonctions cycliques)**.** *Une fonction* $f : \Sigma^* \to \Gamma^*$ *est* cyclique *si* $\forall u \in \Sigma^*, \forall k \geq 1$ *:*
- $f(\sigma^k(u)) = \sigma^k(f(u))$,
- $f(u^k) = f(u)^k$,
- $|f(u)| = |u|$

Remarquons que la première condition impose que $f(u) = \sigma^{|u|}(f(u))$, c'est-à-dire que $f(u)$ soit périodique de période $|u|$. À la lumière de cette observation, notre troisième condition, qui n'est pas à proprement parler nécessaire, apparaît comme une simplification qui ne cause pas de perte de généralité.

3.2.3 Classes de complexité et équivalences

Définition 3.12 (taille d'une entrée)**.** *La* taille N *de l'entrée périodique* $\mathcal{C}_u$ *d'un automate cellulaire cyclique est la longueur de la période minimale de* $\mathcal{C}_u$*.*

Notre modèle de calcul peut également être considéré comme un automate agissant sur un anneau de taille finie. Cette vision du modèle nous permet de nous rendre compte qu'on ne peut utiliser qu'un espace de calcul linéaire en la taille de l'entrée. Par conséquent, tous les langages et les fonctions étudiées doivent appartenir à LINSPACE, la classe usuelle et robuste de l'espace linéaire [40].

Définition 3.13 (classes de complexité)**.**
- $\text{LINSPACE}_{\text{cycle}}(\text{POLY}(N))$ *est la classe de complexité des langages fortement cycle-reconnaissables en temps polynomial.*
- $\text{LINSPACE}(N, \text{POLY}(N))$ *est la classe des langages reconnaissables en espace linéaire et en temps polynomial sur les automates cellulaires à entrée bornée (ou, de manière équivalente, sur d'autres modèles classiques comme les machines de Turing ou les machines RAM).*

Ces notations seront abusivement utilisées pour dénoter les classes de complexité de fonctions correspondantes.

Nous allons prouver dans cette section les résultats suivant :

Théorème 3.14 (B. [2])**.** *Un langage cyclique est fortement cycle-reconnaissable si et seulement si il est faiblement cycle-reconnaissable.*

Théorème 3.15 (B. [2])**.** *Soit* L *un langage cyclique, alors :*
- L *est fortement cycle-reconnaissable* $\iff L \in \text{LINSPACE}$,
- $L \in \text{LINSPACE}_{\text{cycle}}(\text{POLY}(N)) \iff L \in \text{LINSPACE}(N, \text{POLY}(N))$.

Théorème 3.16 (B. [2])**.** *Soit* f *une fonction cyclique, alors :*
- f *est cycle-calculable* $\iff f \in \text{LINSPACE}$,
- $f \in \text{LINSPACE}_{\text{cycle}}(\text{POLY}(N)) \iff f \in \text{LINSPACE}(N, \text{POLY}(N))$.

3.3 Preuve des résultats d'équivalence

Nous allons prouver dans cette section les trois théorèmes énoncés précédemment. Pour ceci, nous démontrerons les implications suivantes :

— si L est faiblement cycle-calculable, alors $L \in$ LINSPACE ;
— si $L \in$ LINSPACE, alors L est faiblement cycle-calculable ;
— si $L \in$ LINSPACE, alors L est fortement cycle-calculable ;
— si $f \in$ LINSPACE, alors f est cycle-calculable ;

De plus, nous effectuerons à chaque étape quelques analyses de complexité qui nous permettront d'obtenir les résultats d'équivalence des classes polynomiales. Nous ne traiterons pas la preuve triviale du fait que $f \in$ LINSPACE si f est cycle-calculable.

3.3.1 Du modèle cyclique vers le modèle standard

Jusqu'à la fin du chapitre, $L \subset \Sigma^*$ dénotera un langage cyclique. Nous montrons dans cette section que si L est faiblement cycle-reconnaissable, alors $L \in$ LINSPACE.

Cette implication est simple : nous allons considérer un automate cellulaire cyclique $\mathcal{A}_{\text{cycle}}$ qui reconnaît faiblement L, et nous en servir pour concevoir un automate cellulaire standard $\mathcal{A}_{\text{borne}}$ (comprendre : à entrée bornée) qui reconnaît L. La construction de $\mathcal{A}_{\text{borne}}$ se déroule comme suit : dans une phase préliminaire, nous allons ajouter une seconde couche de calcul à l'automate, et recopier le mot d'entrée de $\mathcal{A}_{\text{borne}}$ à l'envers dans cette couche. Ensuite, l'automate va simuler le calcul de $\mathcal{A}_{\text{cycle}}$ sur le mot formé par la connexion aux extrémités du mot d'entrée et de son image inversée (voir figure 3.6). Après une phase préliminaire qui se déroule en temps linéaire, $\mathcal{A}_{\text{borne}}$ peut simuler $\mathcal{A}_{\text{cycle}}$ pas à pas.

Toutefois, il faut être prudent lors de la définition des conditions d'arrêt de $\mathcal{A}_{\text{borne}}$. En effet, il peut arriver à un instant donné que toutes les cellules simulées de $\mathcal{A}_{\text{cycle}}$ soient dans le sous-ensemble d'états d'« acceptation » ou de « rejet », mais que le calcul ne soit pas terminé pour autant. Nous devons donc ajouter une couche de calcul supplémentaire à $\mathcal{A}_{\text{borne}}$ qui construit le temps exponentiel après lequel nous sommes assurés que le calcul de $\mathcal{A}_{\text{cycle}}$ est terminé (voir fait 3.9 page 28), puis en décidant si le mot est accepté ou refusé en vérifiant l'état de n'importe laquelle des cellules simulées de $\mathcal{A}_{\text{cycle}}$ (par construction, toutes ces cellules sont dans le même sous-ensemble « acceptation » ou « rejet » à cet instant).

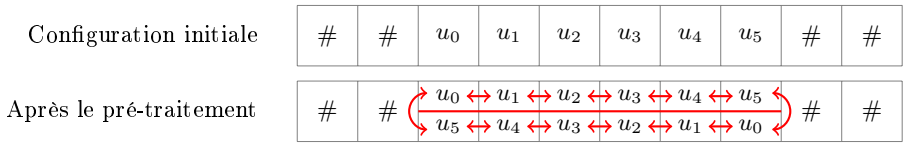


FIGURE 3.6 – Simulation d'un automate cellulaire cyclique sur un automate cellulaire standard.

Il est possible de traiter immédiatement la preuve de la proposition

$$L \in \text{LINSPACE}_{\text{cycle}}(\text{POLY}(N)) \implies L \in \text{LINSPACE}(N, \text{POLY}(N))$$

Il suffit en effet de construire une borne supérieure polynomiale après laquelle le calcul de $\mathcal{A}_{\text{cycle}}$ est terminé (cette borne existe par définition) au lieu de la borne exponentielle précédente, et la même construction fonctionne, toutes choses égales par ailleurs.

3.3.2 Du modèle standard vers le modèle cyclique

Dans cette partie, nous allons prouver que $L \in \text{LINSPACE}$ implique que L est faiblement cycle-reconnaissable.

Soit $\mathcal{A}_{\text{borne}}$ un automate cellulaire à entrée bornée qui reconnaît le langage $L \subset \Sigma^*$ en temps linéaire. Nous allons construire un automate cellulaire cyclique $\mathcal{A}_{\text{cycle}}$ qui va simuler le calcul de $\mathcal{A}_{\text{borne}}$. Si l'ensemble des états de $\mathcal{A}_{\text{borne}}$ est noté $\mathcal{Q}$, notre automate $\mathcal{A}_{\text{cycle}}$ va utiliser un nouvel ensemble d'états $\mathcal{Q}' = \Sigma \times \mathcal{Q} \times \omega$, où ω nous permettra de gérer la simulation de $\mathcal{A}_{\text{borne}}$. Nous souhaitons que chaque cellule de $\mathcal{A}_{\text{cycle}}$ conserve à tout instant le symbole qui lui a été donné en entrée, par conséquent nous allons faire en sorte que la projection de $\mathcal{Q}'$ sur Σ reste constante au cours du temps. Tout le calcul effectif se fera dans les deux autres couches de calcul, à savoir $\mathcal{Q} \times \omega$.

Nous introduisons également une bijection $val : \Sigma \cup \{\vdash\} \to [0, |\Sigma|]$ qui va associer une valeur entière à chaque symbole de Σ, plus une valeur particulière pour le symbole spécial $\vdash$.

Vision globale

Nous allons maintenant présenter un algorithme qui, partant d'une configuration périodique, la partitionne en périodes minimales. Nous utiliserons ensuite cet algorithme pour simuler l'exécution de $\mathcal{A}_{\text{borne}}$ sur le mot contenu dans chaque période minimale. Nous savons que le langage L est cyclique, par conséquent le résultat de l'exécution de $\mathcal{A}_{\text{borne}}$ sur cette période minimale sera le même que ce qu'il aurait été sur le mot qui a généré la configuration périodique initiale.

La configuration initiale de $\mathcal{A}_{\text{cycle}}$ va être découpée en blocs de cellules contiguës qui seront appelés *intervalles*, dans lesquels des exécutions de $\mathcal{A}_{\text{borne}}$ seront simulées. Nous allons concevoir ces intervalles de telle sorte que deux intervalles adjacents contenant des mots différents fusionnent après un certain temps. Pour une compréhension intuitive de l'algorithme, le lecteur pourra imaginer que chaque intervalle peut se trouver dans trois états, à savoir « *fusion à droite* », « *fusion à gauche* » et « *transit* ». Ces états vont évoluer au cours du temps selon un principe qui sera détaillé plus bas ; les intervalles vont fusionner selon la règle suivante :

Règle 1 (Évolution globale)**.** *Si un intervalle est dans l'état «* fusion à droite *» et que son voisin de droite est dans l'état «* fusion à gauche *», alors ils fusionnent l'un avec l'autre.*

Au début de la simulation, chaque cellule va former à elle seule son propre intervalle, et ces intervalles vont fusionner au fur et à mesure des calculs. Nous

affirmons que le processus de fusion va se poursuivre jusqu'à ce que la configuration soit découpée en intervalles identiques (voir la figure 3.7 pour une intuition du processus de fusion).

Une fois que ce découpage est atteint, la simulation de $\mathcal{A}_{\text{borne}}$ dans chaque intervalle et la propagation de l'état d'acceptation/rejet va correspondre à notre critère de reconnaissance faible pour les langages cycliques. Nous verrons plus tard comment améliorer la construction pour obtenir la reconnaissance forte.

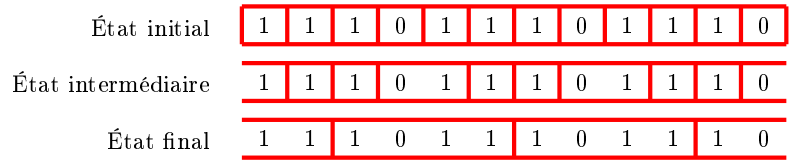


FIGURE 3.7 – Vue d'ensemble du processus de fusion sur une configuration cyclique.

Outils de base

Nous identifions une sous-couche particulière de notre alphabet de travail ω, qui sera de la forme $\omega = \omega' \times \{\#, \varnothing\}$. Cette sous-couche va nous permettre de définir formellement les intervalles.

Définition 3.17 (intervalles)**.** *Les* intervalles *d'une configuration donnée sont les ensemble maximaux (pour l'inclusion) de cellules adjacentes commençant par un # et contenant exactement un seul symbole #. La taille n d'un intervalle I est le nombre de ses cellules.*

La figure 3.8 présente une configuration périodique découpée en intervalles de différentes tailles.

FIGURE 3.8 – Intervalles de tailles 2, 5 et 3.

Définition 3.18 (contenu d'un intervalle)**.** *Le* contenu *d'un intervalle I est le mot résultant de la projection sur Σ des états de ses cellules, précédé du symbole spécial $\vdash$ (nous supposons que $\vdash \notin \Sigma$). Pour un intervalle de taille n, notons a_i le i-ème symbole de son contenu, avec $a_0 = \vdash$.*

Deux intervalles sont dits différents *si leurs contenus sont deux mots différents.*

Nous démontrerons plus tard dans ce chapitre que l'algorithme que nous allons présenter obéit à la propriété suivante :

Lemme 3.19. *Il existe une constante C telle que, si deux intervalles adjacents ont des contenus différents à un instant t, alors au moins l'un d'entre eux aura fusionné avant l'instant $t + C \times n^3$, où n est la taille du plus grand des deux intervalles.*

Signaux et pointeurs Chaque intervalle va maintenir un signal qui effectuera des allers et retours sur son contenu, en commençant par le bord gauche ; les intervalles vont également maintenir un pointeur sur leur contenu (voir figure 3.9). À chaque fois que le signal entrera par la droite dans une cellule contenant le pointeur, il sera déplacé vers la lettre suivante du contenu (à cette fin, nous supposons que le symbole $\vdash$ est stocké dans la première cellule de l'intervalle). Quand le pointeur se trouve sur le dernier symbole du contenu, son prochain mouvement sera de revenir au début (en $n-1$ unités de temps). La position du signal sur le contenu de l'intervalle sera notée i, de telle sorte que a_i soit le symbole actuellement pointé dans l'intervalle.

À l'initialisation de l'algorithme, un symbole # est inscrit dans chaque cellule, de telle sorte que chacune d'entre elles forme un intervalle de taille 1. De plus, un signal est lancé dans chaque intervalle, et tous les pointeurs sont positionnés sur le symbole $\vdash$. Nous remarquons qu'à tout instant une cellule qui contient le symbole # contient un symbole $\vdash$ ainsi que le premier symbole a_0 du contenu de son intervalle.

Notations 3.20 (k,t_i)**.**
Posons

$$k = |\Sigma \cup \{\vdash\}| + 1 = |\Sigma| + 2$$

et

$$\forall i \in [\![0, n]\!] \quad t_i = kn^2 + 2n \times val(a_i)$$

Remarquons que pour tout a_i, $val(a_i) \leq k-1$.

Comportement du signal Pour chaque symbole a_i, le signal va attendre un temps t_i sur le bord gauche de l'intervalle, puis va se déplacer à vitesse maximale jusqu'au bord droit. Il va ensuite attendre le même temps t_i sur le bord droit, puis va à nouveau se déplacer à vitesse maximale vers le bord gauche, en modifiant au passage le symbole a_i pointé dans le mot de contenu. Cette famille de temps $(t_i)_{i\in[0,n]}$ va nous servir à encoder le contenu de l'intervalle. Grâce à cette méthode, deux intervalles voisins pourront comparer leurs contenus, et fusionner ensemble le cas échéant.

Notons que la fonction $n \mapsto kn^2 + 2n \times val(a_i)$ est constructible en temps grâce à des techniques standard (voir [34]), en utilisant des signaux encodant $val(a_i)$ dans leurs états. Cette constructibilité nous permet d'attendre effectivement un temps t_i sur chaque bord.

Revenons à notre vision globale de l'algorithme en considérant qu'un intervalle est dans l'état « *fusion à droite* » (resp. « *fusion à gauche* ») lorsque son signal attend sur le bord droit (resp. sur le bord gauche), et qu'il est dans l'état « *transit* » dans les autres cas.

Processus de fusion

Soient deux intervalles adjacents I_1 et I_2, I_1 étant à gauche de I_2. Considérons un instant où les signaux respectifs de I_1 et I_2 se rencontrent sur leur bord commun. Selon nos considérations précédentes, I_1 est dans l'état « *fusion à droite* », et I_2 est dans l'état « *fusion à gauche* ». Par conséquent, selon la règle 1, I_1 et I_2 doivent fusionner. Cette fusion s'effectue en effaçant le symbole # au début de I_2 qui définit leur bord commun, et en supprimant leurs signaux.

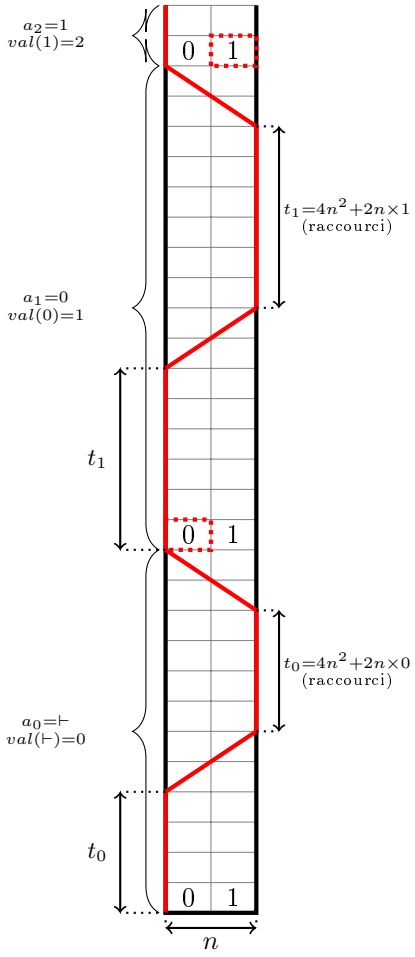


FIGURE 3.9 – Mouvement du signal dans un intervalle de taille 2 (avec $|\Sigma| = 2$). Le carré pointillé symbolise le pointeur sur le contenu, qui est positionné sur ⊢ jusqu'à mention contraire.

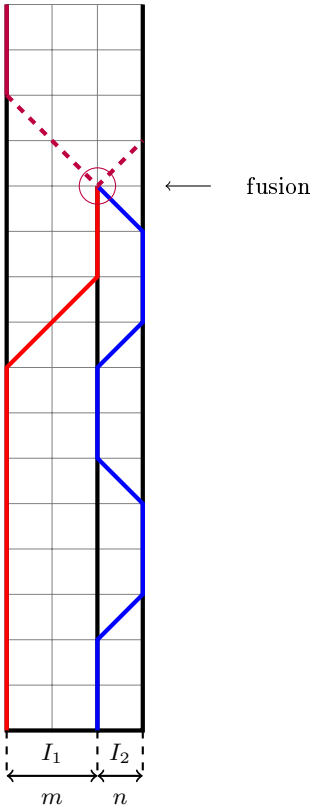


FIGURE 3.10 – Illustration du processus de fusion pour des intervalles de tailles différentes.

Deux signaux sont ensuite envoyés vers le début et la fin de l'intervalle nouvellement créé, pour repositionner le pointeur au début du nouveau contenu, et réinitialiser la simulation de $\mathcal{A}_{\text{borne}}$. Finalement, un nouveau signal de fusion est créé sur le bord gauche du nouvel intervalle, et le comportement normal reprend (voir figure 3.10).

Notons que la ré-initialisation de la simulation de $\mathcal{A}_{\text{borne}}$ doit se faire de manière « propre » : il faut utiliser un algorithme de ligne de fusiliers pour synchroniser le nouvel intervalle, afin que toutes les cellules du nouvel intervalle commencent la simulation en même temps.

Nous avons maintenant tous les outils nécessaires pour prouver le lemme 3.19 :

Soient I_1 et I_2 deux intervalles adjacents dont les contenus a et b sont différents, et soient n_1, n_2, S_1 et S_2 leurs tailles et leurs signaux de fusion respectifs. Nous affirmons sans perte de généralité que I_1 est situé à gauche de I_2. Supposons que les intervalles I_1 et I_2 ne fusionnent pas avec d'autres intervalles pendant les instants considérés (si cette supposition est fausse, alors le lemme est prouvé). Notre preuve considérera deux cas disjoints : si I_1 et I_2 sont de même taille ou non.

Fusion d'intervalles de tailles différentes Supposons sans perte de généralité que $n_1 > n_2$, en particulier $n_1 \geq n_2 + 1$. Nous allons prouver que lorsque S_1 arrive sur la frontière commune aux deux intervalles, il va nécessairement rencontrer S_2 avant de repartir, déclenchant ainsi une fusion.

Nous savons que le signal S_1 reste sur la frontière pendant une durée $T_1 = kn_1^2 + 2n_1 \times val(a_i)$. Nous remarquons immédiatement que $T_1 \geq kn_1^2$.

Considérons maintenant le temps maximal T_2 pendant lequel S_2 *peut être absent* de la frontière commune. Ce temps est la somme du temps maximal pendant lequel S_2 peut attendre sur l'autre frontière de l'intervalle I_2 et du temps nécessaire pour effectuer un aller et retour dans l'intervalle (soit $2n_2$). Nous avons donc :

$$\begin{gathered} T_2 = kn_2^2 + 2n_2 \times \max_j(val(b_j)) + 2n_2 \\ T_2 = kn_2^2 + 2n_2 \times (k-1) + 2n_2 \\ T_2 = kn_2^2 + 2kn_2. \end{gathered}$$

Or $n_1 \geq n_2 + 1$, par conséquent :

$$\begin{gathered} T_1 \geq kn_1^2 \\ T_1 \geq k(n_2+1)^2 \\ T_1 \geq kn_2^2 + 2kn_2 + k \\ T_1 \geq T_2 + k \\ T_1 > T_2. \end{gathered}$$

Nous avons donc prouvé que S_2 allait revenir sur la frontière commune avant que S_1 n'en parte, déclenchant ainsi une fusion entre I_1 et I_2.

Complexité temporelle : Nous remarquons aisément que I_1 et I_2 vont fusionner en temps $O(max(n_1, n_2)^2)$, car le signal du plus grand intervalle ne peut pas accomplir un aller et retour sans rencontrer le signal du plus petit. *A fortiori*, il existe une constante C telle que I_1 et I_2 vont fusionner avant l'instant $t + C \times max(n_1, n_2)^3$, ce qui prouve le lemme 3.19 pour le cas $n_1 \neq n_2$.

Fusion d'intervalles de mêmes tailles Considérons maintenant le cas où I_1 et I_2 ont la même taille n, mais des contenus a et b différents. Nous allons montrer qu'ils fusionnent peu de temps après que leurs pointeurs aient été positionnés sur des symboles différents. Commençons par introduire quelques notations, pour la compréhension desquelles le lecteur est encouragé à se référer à la figure 3.11.

Notations 3.21 (t_1, t'_1, t_2)**.** *Soit t_1 un instant où le signal S_1 atteint le bord gauche de I_1 après avoir effectué un aller et retour dans l'intervalle. Avant d'arriver sur ce bord à l'instant t_1, le signal S_1 a quitté la frontière commune aux deux intervalles à l'instant $t'_1 = t_1 - n$. Notons t_2 le premier instant suivant t'_1 où S_2 atteint la frontière commune.*

Définition 3.22 (asynchronicité)**.** *L'*asynchronicité *entre les intervalles I_1 et I_2 au temps t_1 est la valeur $\delta = t_1 - t_2$.*

Notons que l'asynchronicité entre deux intervalles est définie à chaque fois que S_1 atteint le bord gauche de son intervalle. Ces instants sont particuliers dans notre construction : ce sont les instants où les intervalles modifient le pointeur sur leur contenu. Ainsi, on peut associer à chaque paire d'instants (t_1, t_2) la paire de symboles $(a_i, b_j) \in \{\Sigma \cup \{\vdash\}\}^2$ qui vient d'être nouvellement pointée dans le contenu des intervalles.

Nous remarquons maintenant quelques propriétés sur les bornes de l'asynchronicité :

Lemme 3.23 (bornes de l'asynchronicité)**.** *L'asynchronicité vérifie l'inégalité $-n < \delta < n$ si $a_{i-1} = b_{j-1}$.*

Démonstration. Par construction, il est impossible d'avoir $\delta \geq n$, puisque $t'_1 < t_2$. D'autre part, si δ est l'asynchronicité entre I_1 et I_2 au temps t_1, et si nous supposons que les symboles a_{i-1} et b_{j-1} sont les mêmes (ce qui signifie que S_1 et S_2 ont attendu le même temps sur le bord droit de leur intervalle avant les temps t_1 et t_2 respectivement), alors il est impossible d'avoir $\delta \leq -n$ (cette remarque est illustrée sur la figure 3.12). Le cas contraire signifierait que S_1 et S_2 se sont rencontrés avant les instants t_1 et t_2. □

Remarquons qu'il existe un point dans la suite des (a_i, b_j) définie précédemment où les symboles a_i et b_j sont différents. En effet, le cas contraire signifierait que la suite coïncide notamment sur les symboles $\vdash$, et sur tous les symboles situés entre deux occurrences consécutives du symbole $\vdash$. Rappelons nous que le symbole $\vdash$ marque le début du contenu d'un intervalle ; par conséquent, si la suite coïncide en permanence cela signifie que les contenus des intervalles sont les mêmes. Nous avons fait l'hypothèse que les contenus a et b des intervalles étaient différents, d'où contradiction.

Considérons donc la première paire d'instants (t_1, t_2) pour laquelle les symboles associés a_i et b_j sont différents, et notons δ l'asynchronicité entre les deux intervalles à l'instant t_1. Puisque $a_{i-1} = b_{j-1}$ par définition, la remarque sur les bornes de l'asynchronicité est valable ; on a donc $|\delta| < n$. Nous affirmons que I_1 et I_2 vont fusionner avant que leurs signaux respectifs n'effectuent un aller et retour. Nous traitons deux cas séparément : le cas $val(a_i) < val(b_j)$ et le cas contraire (voir figure 3.13).

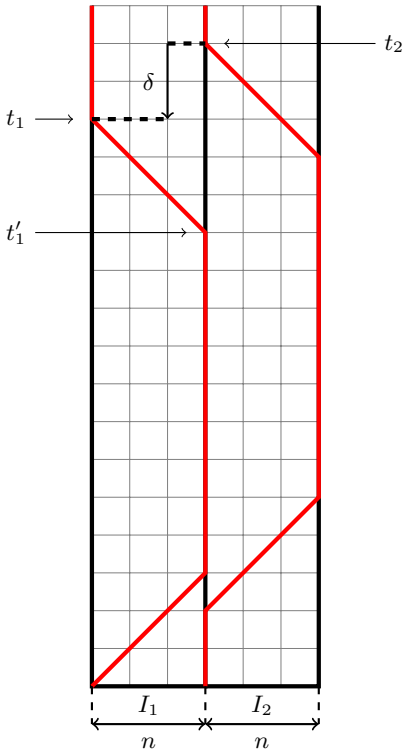


FIGURE 3.11 – Définition de l'asynchronicité entre I_1 et I_2 au temps t_1.

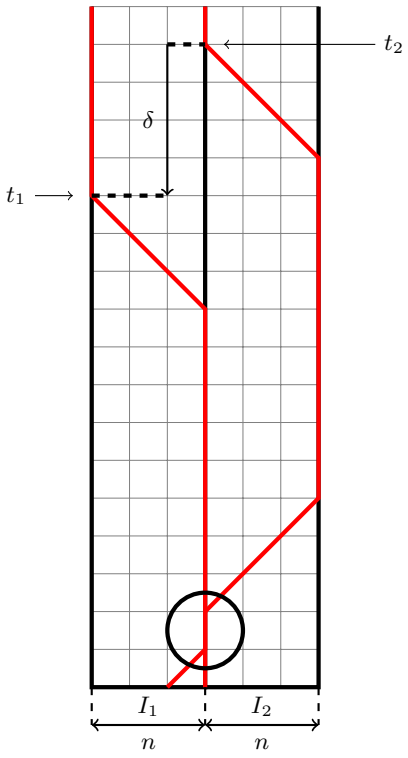


FIGURE 3.12 – Le cas absurde $\delta \leq -n$.

Considérons d'abord le cas $val(a_i) < val(b_j)$ (voir figure 3.13a). S_1 va attendre un délai $T_0 = kn^2 + 2n \times val(a_i)$ sur chaque bord de I_1, et S_2 va attendre un délai $T_0 + \Delta_t$, avec $\Delta_t = 2n \times (val(b_j) - val(a_i))$. Puisque $val(b_j) > val(a_i)$, nous avons donc $\Delta_t \geq 2n$.

S_1 va arriver sur le bord commun aux deux intervalles à l'instant $t_1 + T_0 + n$, et S_2 va rester sur ce même bord de l'instant $t_1 + \delta$ à l'instant $t_1 + \delta + T_0 + \Delta_t$. Nous allons montrer que $T_0 + n \in [\delta, \delta + T_0 + \Delta_t]$, autrement dit que les deux signaux se rencontreront sur la frontière commune à l'instant $t_1 + T_0 + n$.

Nous avons $T_0 > 0$ et $|\delta| < n$, par conséquent $T_0 + n > \delta$. Par ailleurs, $\Delta_t \geq 2n$, donc $\delta + \Delta_t > n$, et finalement $T_0 + n < \delta + T_0 + \Delta_t$, ce qui conclut le cas $val(a_i) < val(b_j)$.

Considérons maintenant le cas $val(a_i) > val(b_j)$ (illustré sur la figure 3.13b) ; nous verrons que la fusion se produira plus tard, mais avec des arguments similaires. Définissons cette fois $T_0 = kn^2 + 2n \times val(b_j)$ comme le délai d'attente de S_2, et $\Delta_t = 2n \times (val(a_i) - val(b_j))$ (pour le confort de la démonstration, nous préférons avoir $\Delta_t > 0$; remarquons que la propriété $\Delta_t \geq 2n$ est toujours vraie). Le délai d'attente de S_1 est donc $T_0 + \Delta_t$. Nous voulons prouver que la fusion se produit au *retour* de S_2 sur la frontière commune, à l'instant $t_1 + \delta + 2(T_0 + n)$ (voir figure 3.13b).

Nous savons que S_1 est présent sur la frontière commune de l'instant $t_1 + T_0 + \Delta_t + n$ à l'instant $t_1 + n + 2(T_0 + \Delta_t)$. Nous devons donc montrer que :

$$t_1 + T_0 + \Delta_t + n \leq t_1 + \delta + 2(T_0 + n) \leq t_1 + n + 2(T_0 + \Delta_t)$$

autrement dit, que

$$\Delta_t \leq \delta + T_0 + n \leq T_0 + 2\Delta_t$$

La borne supérieure est facile à démontrer : il suffit de remarquer que :

$$\delta + n \leq 2n < 4n \leq 2\Delta_t$$

Pour démontrer la borne inférieure, il faut revenir sur l'expression de Δ_t. Nous savons que $\Delta_t = 2n \times (val(a_i) - val(b_j))$, or $\forall a_i \; val(a_i) \in [0, k]$. Nous en déduisons donc que :

$$\Delta_t \leq 2kn \leq kn^2 \leq T_0 \text{ (si } n > 1)$$

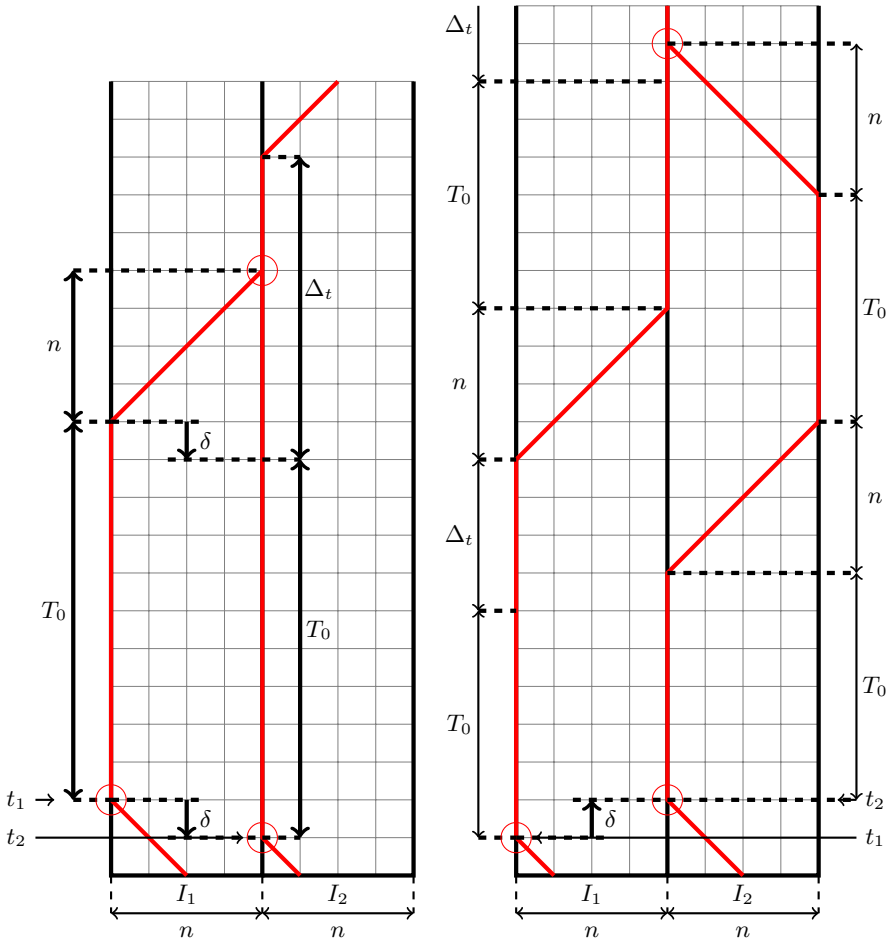
Par ailleurs, nous savons que $\delta + n > 0$, nous en concluons finalement que :

$$\Delta_t \leq T_0 + \delta + n \text{ (si } n > 1)$$

Le cas particulier où les deux intervalles sont distincts et de taille $n = 1$ se règle facilement.

Analyse de complexité : Soit la paire (t_1, t_2) qui spécifie, comme précédemment, la première paire d'instants pour laquelle les symboles associés a_i et b_j sont différents. Nous remarquons que la fusion entre I_1 et I_2 doit s'effectuer dans un délai $O(n^2)$ après l'instant t_1, puisque les signaux ne peuvent pas accomplir un aller et retour dans leurs intervalles sans fusionner à partir de t_1.

Considérons maintenant le temps avant qu'une telle paire soit observée pour les intervalles I_1 et I_2 : nous savons par construction que le symbole pointé dans un intervalle change avec un délai $O(n^2)$. D'autre part, le pointeur sur le contenu de l'intervalle est réinitialisé une fois qu'il est passé par les $n+1$ symboles du contenu du mot. Par conséquent, si une telle paire (t_1, t_2) existe, alors elle apparaît dans un laps de temps $O(n^3)$, ce qui prouve le dernier cas de notre lemme 3.19.



(a) $val(a_i) < val(b_j)$ (b) $val(a_i) > val(b_j)$

FIGURE 3.13 – Illustration du processus de fusion pour des intervalles de même taille.

Rappelons nous maintenant que la configuration est initialement divisée en intervalles de taille 1. Il est évident qu'au cours de l'algorithme la taille des intervalles ne peut être supérieure à la taille N de la période minimale de la configuration, et que le calcul se termine lorsque cette taille est atteinte. Considérons l'ensemble des N intervalles initiaux de taille 1 qui finiront par fusionner pour former un unique intervalle de taille N en fin de calcul. Le lemme 3.19

établit que, tant qu'il existe deux intervalles différents et adjacents dans cet ensemble, alors au moins l'un d'entre eux doit fusionner avant un temps $O(N^3)$. Ainsi, il existe un délai $T_f = CN^3$ tel que le nombre d'intervalles dans l'ensemble considéré décroît strictement tous les T_f, jusqu'à ce que l'ensemble soit un singleton (c'est-à-dire que tous les intervalles considérés ont fusionné). Nous établissons ainsi que l'algorithme de fusion d'intervalles se termine en temps $O(N^4)$.

À partir de ce point, il nous suffit d'attendre la fin de la simulation de $\mathcal{A}_{\text{borne}}$ dans chaque intervalle, et de déterminer l'acceptation ou le rejet de la configuration périodique initiale en observant la projection sur $\mathcal{Q}$ des états de $\mathcal{A}_{\text{cycle}}$. Étant donné que cette projection ne changera plus au cours de l'évolution de l'automate (puisque les intervalles ne fusionneront plus, la simulation de $\mathcal{A}_{\text{borne}}$ ne sera plus interrompue), le comportement de $\mathcal{A}_{\text{cycle}}$ correspond à notre définition de la reconnaissance faible.

La preuve reste la même si nous avons $L \in \text{LINSPACE}(N, \text{POLY}(N))$ et que nous voulons prouver que $L \in \text{LINSPACE}_{\text{cycle}}(\text{POLY}(N))$. En effet, notre construction ne fait qu'ajouter un temps $O(N^4)$ avant que la simulation de $\mathcal{A}_{\text{borne}}$ ne décide en temps polynomial si l'entrée appartient ou non à L.

3.3.3 De la reconnaissance faible vers la reconnaissance forte

Nous allons maintenant raffiner notre construction de telle sorte que l'automate cellulaire cyclique $\mathcal{A}_{\text{cycle}}$ reconnaisse *fortement* le langage L si et seulement si l'automate simulé $\mathcal{A}_{\text{borne}}$ le reconnaît. La première chose à faire est d'ajouter deux états à notre alphabet de travail $\mathcal{Q}'$ pour encoder l'acceptation ou le rejet d'une configuration. Notre nouvel alphabet est donc :

$$\mathcal{Q}' = (\Sigma \times \mathcal{Q} \times \omega) \cup \{acc, rej\}$$

Une manière naïve d'atteindre la reconnaissance forte serait de remplacer les états de toutes les cellules d'un intervalle par acc ou rej lorsque la simulation de $\mathcal{A}_{\text{borne}}$ dans cet intervalle est terminée. C'est la bonne piste à suivre, mais elle mène à quelques problèmes que nous allons détailler et résoudre dans les paragraphes suivants.

Faux positifs Le problème principal auquel nous devons faire face lorsque nous remplaçons le contenu d'une cellule par des états acc ou rej est que l'on supprime les « traits de construction » de l'intervalle, et notamment tous les signaux et pointeurs qu'il contenait. Nous nous trouvons alors face à la possibilité que l'intervalle n'était pas maximal, et que ses voisins aient eu besoin de fusionner avec lui. L'intervalle estimerait donc localement que le calcul est terminé, alors que ce n'est pas le cas globalement. De plus, même si nous parvenions d'une manière ou d'une autre à reconstruire l'intervalle, son contenu initial (un mot de Σ^*) serait perdu. Le mécanisme présenté dans le paragraphe suivant nous permet de résoudre ce problème.

Sauvegarde du contenu Nous ne voulons pas qu'un intervalle ne remplace son contenu par des états acc ou rej tant qu'il n'est pas certain que son contenu puisse être restauré si nécessaire. Nous allons pour cela nous rendre compte

qu'il est possible pour un intervalle de déterminer si un de ses voisins a le même contenu que lui. Le lemme 3.19 nous assure que si deux intervalles adjacents cohabitent pendant un certain temps, alors ils ont le même contenu. Nous voulons que les intervalles détectent ces cas (qui arriveront nécessairement, à moins que l'intervalle ne fusionne), et ne remplacent leur contenu que lorsqu'ils sont assurés que le contenu de leur voisin de gauche est le même que le leur.

Tout d'abord, remarquons qu'il est facile de détecter si le voisin de gauche d'un intervalle est de la même taille que ce dernier : si le voisin est de plus petite taille, nous avons établi précédemment qu'ils vont fusionner lorsque le signal de l'intervalle considéré attendra sur son bord gauche. Si le voisin est de plus grande taille, il suffit de remarquer que deux passages consécutifs du signal sur le bord gauche de son intervalle ont lieu sans que le signal du voisin ne s'y rende (il peut laisser sur son bord gauche un jeton « consommé » par le signal du voisin pour s'en rendre compte).

Dès lors qu'un intervalle sait que son voisin de gauche est de la même taille que lui, le lemme 3.19 nous affirme que si ni un intervalle ni son voisin de gauche n'ont fusionné pendant un temps Cn^3, alors ils ont le même contenu. Nous allons utiliser cette propriété en ajoutant une couche de calcul supplémentaire dans les intervalles qui fera office de « minuteur » qui s'activera dès que la simulation de $\mathcal{A}_{\text{borne}}$ s'achèvera et qui déclenchera le remplacement du contenu de l'intervalle par acc ou rej, à moins que ce minuteur ne soit interrompu par la fusion de l'intervalle ou celle de son voisin de gauche.

Restauration du contenu Puisque le voisin de droite d'un intervalle peut compter sur lui pour restaurer son contenu, nous devons nous assurer qu'à chaque fois qu'un intervalle est modifié (c'est-à-dire lorsqu'il fusionne) son ancien contenu soit envoyé dans son voisin de droite pour créer un nouvel intervalle et remplacer les états acc ou rej qui s'y trouvaient. Une intuition du mécanisme permettant de recopier le contenu d'un intervalle à sa droite est donnée dans la figure 3.14. Le mécanisme complet utilise des signaux à pente variable, et ne sera pas détaillé ici. Lorsqu'un intervalle récupère son contenu par cette méthode, il créé un nouveau signal de fusion, et recommence ses calculs comme un nouvel intervalle résultant d'une fusion ; il vérifie aussi si son propre voisin de droite a besoin d'être restauré, et le restaure par le même mécanisme le cas échéant.

Notre automate ainsi modifié effectue bien la reconnaissance forte du langage reconnu par l'automate $\mathcal{A}_{\text{borne}}$. En effet, lorsque les intervalles sont maximaux, ils détectent tous simultanément que leur voisin de gauche a le même contenu qu'eux (de fait, ils sont eux-même leur propre voisin de gauche, dans un certain sens), puis ils effacent simultanément leur contenu pour le remplacer par l'état acc ou rej. Puisque ces deux états sont quiescents, la fonction globale de transition a bien atteint un point fixe.

3.3.4 Calcul de fonctions

Supposons que nous disposons d'un automate $\mathcal{A}_{\text{borne}}$ qui calcule une fonction cyclique f. Il suffit d'une légère modification au mécanisme qui nous a permis d'obtenir la reconnaissance forte d'un langage pour obtenir un automate cellulaire cyclique qui calcule la fonction f. En effet, au lieu d'ajouter l'ensemble $\{acc, rej\}$ à $\mathcal{Q}'$, il suffit de lui ajouter Γ, l'alphabet de sortie de la fonction f. Au lieu de remplacer leurs états par acc ou rej, les intervalles écrivent le résultat de

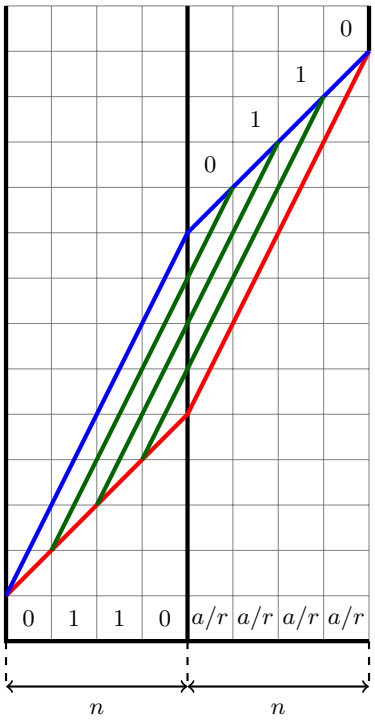


FIGURE 3.14 – Une intuition du mécanisme de copie de contenu.

l'application de la fonction f sur leur contenu (ils ont la place de le faire, puisque $|f(u)| = |u|$). Toutes choses égales par ailleurs, l'automate ainsi modifié calcule f, qui est donc par conséquent cycle-calculable. Les bornes polynomiales que nous avions établies sont toujours valables, par conséquent les deux théorèmes portant sur les classes de complexité de fonctions sont prouvés.

Problème de la densité : Remémorons nous le problème de la densité, ou « Density Classification Problem » tel que défini dans l'article de Land [27] et notamment étudié dans des articles de Fatès [18] et Fúks [19]. Ce problème consiste à concevoir un automate cellulaire cyclique sur l'alphabet $\{0, 1\}$ qui converge sur la configuration uniformément composée de 0 (resp. de 1) si sa configuration initiale contient plus de 0 que de 1 (resp. plus de 1 que de 0). Bien que la résolution de ce problème ait été prouvée impossible dans le cadre originel des automates cellulaires déterministes à deux états [27], plusieurs solutions ont été étudiées en s'autorisant certains relâchements (versions asynchrones, automates probabilistes, etc...).

Nous nous apercevons que la fonction qui associe à une configuration périodique donnée la configuration uniformément composée de 0 ou de 1 selon sa densité est une fonction cyclique. Par conséquent, la construction que nous présentons dans ce chapitre permet de résoudre le problème de la densité en s'autorisant un relâchement qui n'avait pas encore été étudié à notre connaissance : l'augmentation du nombre d'états, tout en conservant un automate cellulaire déterministe.

Il faut toutefois noter que la résolution du problème de la densité que nous proposons présente l'inconvénient d'utiliser un alphabet de sortie différent de l'alphabet d'entrée, ce qui est un relâchement supplémentaire par rapport au problème initial. Dans ce cas précis, il est possible de légèrement modifier l'algorithme pour obtenir une sortie écrite dans le même alphabet que l'entrée, sans craindre que le calcul ne reprenne depuis le début : l'intuition serait d'écrire les symboles # servant à délimiter les intervalles seulement lorsqu'un motif 01 ou 10 est rencontré sur la configuration initiale.

Chapitre 4

Configurations périodiques de dimension 2 : motifs primitifs et élection de leader

Dans le chapitre précédent, nous avons présenté un algorithme qui calcule en temps polynomial la période minimale d'une configuration périodique, en dimension 1. Nous en avons ensuite déduit l'égalité (au temps polynomial près) des classes de complexité des automates cellulaires dans le modèle périodique et dans le modèle standard, pour les langages cycliques.

Dans ce chapitre, nous cherchons à étendre ces résultats à la dimension 2. Nous nous intéressons donc à des automates cellulaires de dimension 2 agissant sur des configurations bi-périodiques, autrement dit des *automates toriques*, qui peuvent être vus comme des automates cellulaires dont la structure sous-jacente est un tore. L'entrée d'un tel automate est une image torique : une image périodique définie à *shift* près sur ses 2 dimensions.

Il n'est pas surprenant de constater que la situation des problèmes posés est plus complexe en dimension 2 qu'en dimension 1. La difficulté réside dans la définition en dimension 2 d'un analogue à la notion de motif primitif d'un mot : en général, un tel motif primitif rectangulaire pave l'image bi-périodique *avec un décalage*, contrairement à un motif primitif.

Nous établissons d'abord que toute image bi-périodique possède plusieurs racines primitives, potentiellement confondues en une seule. Chacun de ces motifs est composé d'exactement un représentant de chaque classe d'équivalence des pixels de l'image, ils possèdent donc tous le même nombre N de pixels. Nous présentons ensuite un algorithme qui identifie une classe d'équivalence de pixels de l'image, appelée « leader », en temps polynomial en N. Cet algorithme est une extension de l'algorithme du chapitre précédent qui calcule la période minimale d'un mot périodique ; la notion d'intervalle en dimension 1 est remplacée par les notions de « patch » et d'arbre de recouvrement d'un patch. Finalement, nous montrons comment découper l'image en motifs primitifs à partir du réseau de $\mathbb{Z}^2$ constitué des pixels de la classe d'équivalence leader.

Enfin, partant du résultat précédent, nous examinons la question de l'extension à la dimension 2 des résultats de complexité obtenus en dimension 1, c'est-à-dire la question de la comparaison des classes de complexité des automates toriques avec celles des automates cellulaires standards de dimension 2. Cette comparaison porterait sur des langages constitués d'images qui sont des motifs primitifs. Malheureusement, nous verrons que de tels langages ne sont pas formellement définis, ce qui nous empêche d'établir les résultats d'équivalence que nous souhaiterions. Toutefois, la construction algorithmique que nous présentons nous permettrait d'établir ces résultats si le problème de la définition d'un « langage bidimensionnel primitif » était résolu.

4.1 Racines primitives

Il y avait un lien fort dans le cas de la dimension 1 entre les mots primitifs et l'élection de leader : élire un leader revient clairement à fragmenter le mot périodique en motifs primitifs. Nous allons nous intéresser dans cette section à une extension possible des mots primitifs en dimension 2, nous appellerons cette extension *racine primitive* d'une image bi-périodique. Il convient de noter que cette extension relativement naturelle du concept de mot primitif n'a à notre connaissance jamais été l'objet d'une étude détaillée. Alors qu'un mot en dimension 1 pouvait être qualifié de *primitif* indépendamment du contexte, on verra ici qu'une racine primitive est nécessairement associée à une image donnée.

4.1.1 Contexte et définitions

Nous allons introduire quelques définitions qui nous mèneront à la définition formelle d'une racine primitive d'une image bidimensionnelle.

Définition 4.1 (image, image bi-périodique)**.** *Soit Σ un alphabet fini, une* image sur Σ *est une fonction $P : \mathbb{Z}^2 \to \Sigma$. Une image est dite* bi-périodique *s'il existe deux vecteurs non colinéaires $(x_0, y_0), (x_1, y_1) \in \mathbb{Z}^2$ appelés* période *de P tels que*

$$\forall (x, y) \in \mathbb{Z}^2 : P(x + x_0, y + y_0) = P(x + x_1, y + y_1) = P(x, y)$$

Dans le contexte des images, un élément $(x, y) \in \mathbb{Z}^2$ est appelé un pixel.

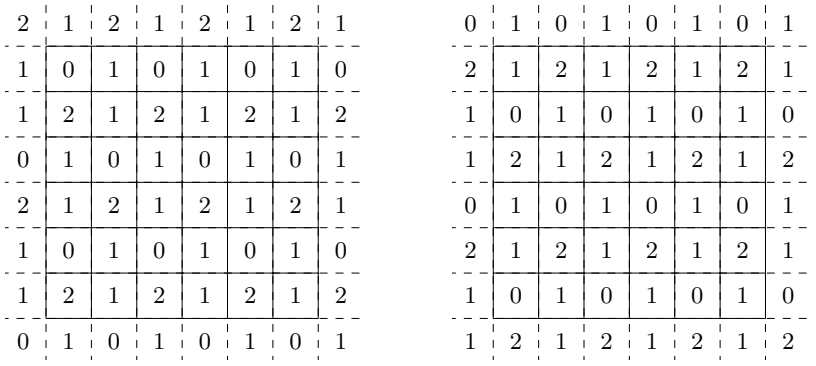
Tout au long de cette section, les images que nous considérerons seront bi-périodiques, sauf si le contraire est explicitement indiqué.

Définition 4.2 (fonctions *shift*)**.** *Les fonctions* shift horizontale σ_h *et* shift verticale σ_v *sont des fonctions définies sur les images de la façon suivante :*

$$\forall (x, y) \in \mathbb{Z}^2 : \sigma_h(P)(x, y) = P(x + 1, y) \text{ et } \sigma_v(P)(x, y) = P(x, y + 1)$$

La figure 4.1 illustre les effets des fonctions *shift* sur une image bi-périodique. Il est facile de voir que ces fonctions *shift* sont inversibles, et qu'elles commutent l'une avec l'autre.

Définition 4.3 (pixels équivalents)**.** *Deux pixels $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$ d'une image P sont dits* équivalents *si la translation qui amène p_1 sur p_2 laisse l'image inchangée, c'est à dire si $\sigma_h^{x_2 - x_1} \circ \sigma_v^{y_2 - y_1}(P) = P$. On note alors $p_1 \sim p_2$.*



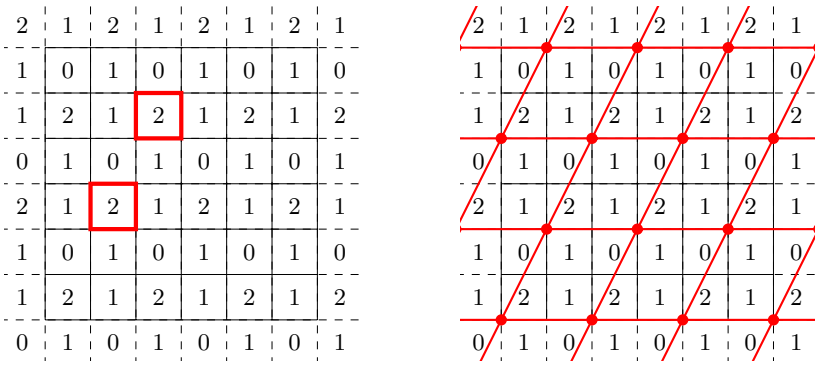
(a) L'image originale P

(b) La nouvelle image $\sigma_h^{-1} \circ \sigma_v^{-1}(P)$

FIGURE 4.1 – Illustration de l'effet des fonctions *shift* sur une image.

On note que cette définition correspond à une relation d'équivalence. Nous en déduisons la propriété suivante sur les classes d'équivalence des pixels :

Fait 4.4. *Pour toute image bi-périodique P, il existe un nombre fini de classes d'équivalence de ses pixels. De plus, chaque classe d'équivalence contient un nombre infini de pixels. Enfin, la classe d'équivalence du pixel $(0,0)$ constitue un* réseau entier de dimension 2, *c'est à dire un sous-groupe de $(\mathbb{Z}^2, +)$ (voir figure 4.2b).*



(a) Deux pixels équivalents.

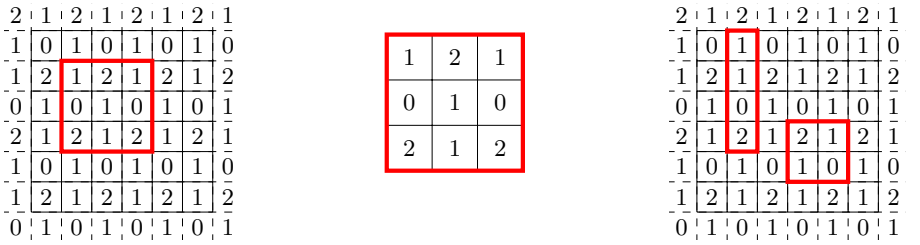
(b) Le réseau associé au pixel $(0,0)$.

FIGURE 4.2 – Les pixels équivalents et le réseau qu'ils induisent.

Nous voulons maintenant pouvoir considérer des sous-parties finies rectangulaires de notre configuration bi-périodique. C'est le sens de notre prochaine définition.

Définition 4.5 (motifs rectangulaires)**.** *Soit P une image sur Σ, le* motif rectangulaire *de taille $m \times n$ extrait en (x_0, y_0) est la fonction suivante :*

$$\begin{array}{rccc} R_{x_0,y_0} & : & [\![0, m-1]\!] \times [\![0, n-1]\!] & \to & \Sigma \\ & & (x, y) & \mapsto & P(x + x_0, y + y_0) \end{array}$$



(a) L'image originale P

(b) Le motif rectangulaire de taille 3×3 extrait en $(1, 2)$

(c) Des racines primitives de l'image P

FIGURE 4.3 – Motifs rectangulaires d'une image bi-périodique.

Définition 4.6 (racines primitives)**.** *Un motif rectangulaire R_{x_0,y_0} de taille $m \times n$ est une* racine primitive *de l'image P s'il contient exactement un représentant de chaque classe d'équivalence de P, c'est à dire si :*
$\forall (x, y) \in \mathbb{Z}^2;\ \exists !(x', y') \in [\![0, m-1]\!] \times [\![0, n-1]\!]$ *tel que* $(x, y) \sim (x_0 + x', y_0 + y')$.

La figure 4.3 donne un exemple de racines primitives d'une image.
Nous déduisons facilement de la définition précédente le fait suivant :

Fait 4.7. *Toutes les racines primitives d'une image donnée ont la même surface, qui est le nombre de classes d'équivalence des pixels de cette image.*

Remarques : Cette définition non constructive est un analogue pour la dimension 2 de la notion de mot primitif dans le cas de la dimension 1. Un premier résultat non trivial est alors que les racines primitives d'une image existent.

Théorème 4.8 (B. [4])**.** *Soit P une image bi-périodique, alors P admet des racines primitives.*

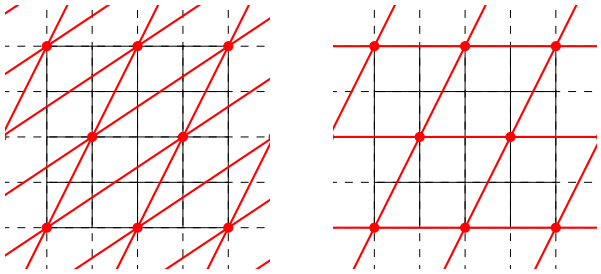
Preuve du théorème 4.8. Cette preuve utilise la notion de *forme normale de Hermite* d'une matrice carrée, qui est un outil bien étudié d'algèbre linéaire. Pour résumer, on dit qu'une matrice entière H est sous forme normale de Hermite si :

— elle est triangulaire inférieure,
— ses coefficients diagonaux sont positifs,
— dans chaque colonne, les coefficients sous la diagonale sont positifs ou nuls, et inférieurs au coefficient de la diagonale.

Pour toute matrice entière M, il existe une matrice H sous forme normale de Hermite telle que $H = U \times M$, où U est une matrice unimodulaire (son déterminant vaut ± 1) à coefficients entiers.

Nous allons aussi utiliser des notions relatives aux *réseaux entiers*, telles que la notion de *domaine fondamental* d'un réseau. Nous n'allons pas prouver dans cette thèse les propriétés des formes normales de Hermite ou des domaines fondamentaux des réseaux, toutefois nous nous permettrons d'utiliser ces propriétés de manière intuitive dans cette preuve. Davantage d'informations sur la forme normale de Hermite et les réseaux peuvent être notamment trouvées dans le cours de Cohen [12].

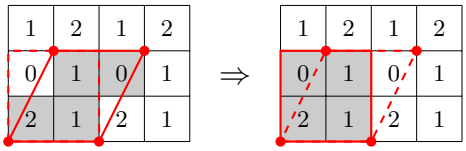
Considérons le réseau $\mathcal{L}$ formé par la classe d'équivalence du pixel $(0, 0)$ (voir fait 4.4). Soit $\mathcal{B}$ une base de ce réseau (voir figure 4.4a), et soit $\mathcal{B}'$ la famille



$\mathcal{B} = \begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix}$ $\mathcal{B}' = \begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}$

(a) Le réseau original $\mathcal{L}$ avec une base quelconque

(b) Le même réseau, avec une base en forme normale de Hermite



(c) Transformation du domaine fondamental $\mathcal{F}$ de $\mathcal{B}'$ en un motif rectangulaire R

FIGURE 4.4 – Illustration de la construction d'une racine primitive.

de vecteurs dont la matrice est la forme normale de Hermite de la matrice associée à $\mathcal{B}$ (voir [12]). Sans perdre de généralité, nous pouvons supposer que $\mathcal{B}' = \begin{pmatrix} \alpha & 0 \\ \beta & \gamma \end{pmatrix}$.
À cause des propriétés de la transformation de Hermite (en particulier, parce que la matrice U est unimodulaire), il est clair que la famille de vecteurs $\{(\alpha, 0), (\beta, \gamma)\}$ est également une base de $\mathcal{L}$ (voir figure 4.4b).

Considérons maintenant $\mathcal{F}$, le domaine fondamental de $\mathcal{L}$ associé à la base $\mathcal{B}'$ (tel que défini dans [12] et représenté en gris dans la figure 4.4c). Plus précisément, considérons les pixels à l'intérieur du domaine $\mathcal{F}$. Puisque $\mathcal{B}'$ est une base de $\mathcal{L}$, il doit nécessairement y avoir exactement un représentant de chaque classe d'équivalence parmi eux (voir figure 4.4c).

Finalement, soit R le motif rectangulaire de taille $\alpha \times \gamma$ extrait de P en position $(0, 0)$ (voir figure 4.4c). Il apparaît que R contient exactement les mêmes classes d'équivalence que $\mathcal{F}$, car tout pixel de R est soit un pixel de $\mathcal{F}$, soit la translation par un vecteur $(-\alpha, 0)$ d'un pixel de $\mathcal{F}$ (une telle translation préserve les classes d'équivalence, puisque $(\alpha, 0)$ est un vecteur de $\mathcal{B}'$).

Nous avons donc montré que le motif rectangulaire R contient exactement un représentant de chaque classe d'équivalence, ce qui fait donc de R une racine primitive. □

On remarque que la construction d'une racine primitive n'est pas triviale dans le cas général : la méthode naïve, qui consisterait à prendre un motif rectangulaire contenant au moins (resp. au plus) un représentant de chaque classe d'équivalence et à le rétrécir (resp. l'agrandir) jusqu'à ce qu'il contienne

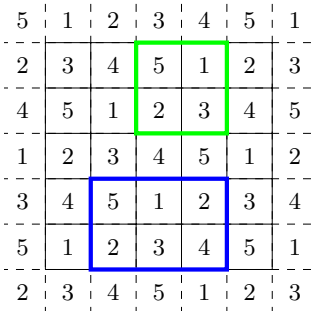


FIGURE 4.5 – Contre-exemples à la méthode naïve.

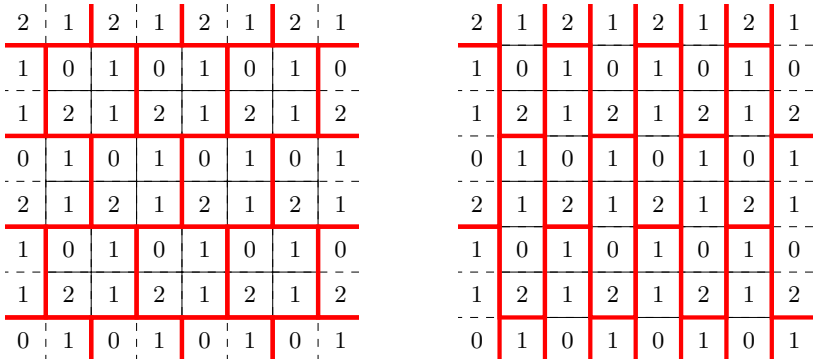


FIGURE 4.6 – Pavages d'une image bi-périodique par ses racines primitives.

exactement un représentant de chaque classe, ne fonctionne pas dans tous les cas. Ce fait est illustré par la figure 4.5, qui donne des exemples de motifs rectangulaires qui ne sont pas des racines primitives, qui contiennent au moins (resp. au plus) un représentant de chaque classe d'équivalence, et qui ne peuvent pas être rétréci (resp. agrandi) en conservant cette propriété.

4.1.2 Caractérisation des racines primitives

Maintenant que nous avons prouvé l'existence des racines primitives dans toute image bi-périodique, nous allons en proposer une caractérisation exacte. Pour cela, nous allons utiliser le lemme suivant :

Lemme 4.9. *Soit R une racine primitive d'une image bi-périodique P, alors R pave P par translation.*

Une illustration de ce lemme est présentée sur la figure 4.6. Le lecteur pourra se convaincre de sa véracité en se rendant compte qu'il est possible de construire une translation de R autour de chaque pixel de P, car R contient un représentant de chaque classe d'équivalence. De plus, ces translations ne peuvent pas se superposer, car cela signifierait qu'une classe d'équivalence possède au moins 2 représentants dans R, ce qui est impossible par construction. Ce pavage peut être obtenu en copiant R en chaque point du réseau $\mathcal{L}$ défini précédemment.

Nous pouvons maintenant introduire le théorème suivant, qui nous donne une caractérisation exacte des racines primitives d'une image.

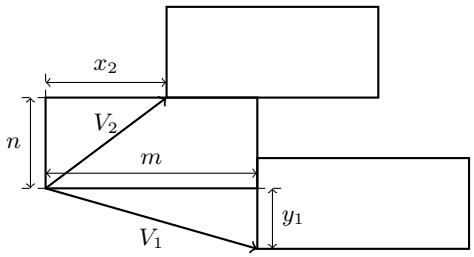


FIGURE 4.7 – Les vecteurs de répétition associés à une racine primitive donnée.

Théorème 4.10 (B. [4]). *Soit P une image bi-périodique, et soit $S \subset \mathbb{N}^2$ l'ensemble de toutes les tailles possibles des racines primitives de P (formellement, $S = \{(m,n);\ \exists R_{x,y}$ une racine primitive de P de taille $m \times n\}$), alors :*

- *les racines primitives d'une image peuvent avoir au maximum deux tailles différentes : $\|S\| \leq 2$;*
- *une racine primitive peut être extraite de n'importe quel point de P, pourvu qu'elle soit de dimension appropriée : $\forall (m,n) \in S;\ \forall (x,y) \in \mathbb{Z}^2$, si $R_{x,y}$ est le motif rectangulaire de taille $m \times n$ extrait de P en (x,y), alors $R_{x,y}$ est une racine primitive de P.*

Preuve du théorème 4.10. Nous allons prouver le premier point du théorème 4.10 en associant une matrice sous forme normale de Hermite à chaque racine primitive d'une image P, et en considérant les implications qui en découlent.

Soit R_{x_0,y_0} une racine primitive de P de taille $m \times n$. Grâce au lemme 4.9, nous savons qu'il existe un pavage de P par R. Considérons maintenant deux vecteurs de répétition particuliers de ce pavage, qui sont illustrés sur la figure 4.7 et qui sont définis comme suit :

- $V_1 = (m, -y_1)$ où y_1 est le plus petit entier positif tel que $(x_0, y_0) \sim (x_0 + m, y_0 - y_1)$.
- $V_2 = (x_2, n)$ où x_2 est le plus petit entier positif tel que $(x_0, y_0) \sim (x_0 + x_2, y_0 + n)$.

De manière intuitive, V_1 et V_2 sont les plus petits (en un certain sens) vecteurs permettant de reconstruire le pavage de l'image par R.

Puisque R pave l'image, il est clair que $0 \leq y_1 < n$ et $0 \leq x_2 < m$. Nous allons maintenant prouver qu'au moins une de ces affirmations est vraie : $y_1 = 0$ ou $x_2 = 0$.

En effet, si $y_1 \neq 0$ et $x_2 \neq 0$, cela signifierait qu'il existe un « trou » rectangulaire de taille $x_2 \times y_1$ dans le pavage (voir figure 4.8). Puisque $x_2 < m$ et $y_1 < n$, cela signifie qu'il est impossible de remplir ce trou avec une translation de R. Il y a donc une contradiction avec le fait que R pave l'image. Il est donc nécessaire que $V_1 = (m, 0)$, ou bien que $V_2 = (0, n)$.

Notons que les vecteurs V_1 et V_2 constituent une base du réseau $\mathcal{L}$ associé à P (puisqu'ils sont non-colinéaires). À un ré-ordonnancement près des dimensions, nous pouvons supposer sans perdre de généralité que $V_1 = (m, 0)$ et $V_2 = (x_2, n)$. Considérons maintenant la matrice $\mathcal{B} = \begin{pmatrix} m & 0 \\ x_2 & n \end{pmatrix}$.

Il s'agit de la matrice d'une base de $\mathcal{L}$, et nous pouvons remarquer qu'elle est sous forme normale de Hermite (puisque $0 \leq x_2 < m$). Cela signifie que tout couple (m, n) éligible pour être la dimension d'une racine primitive de P doit être

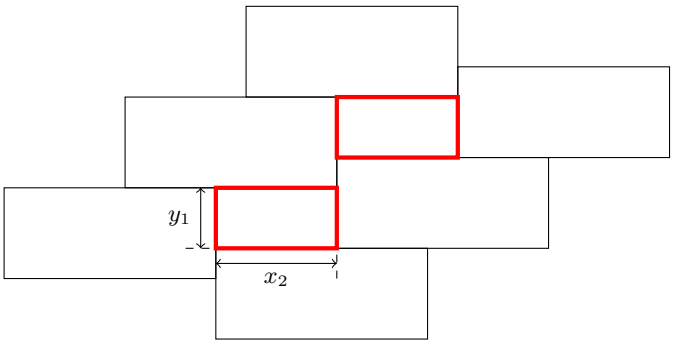


FIGURE 4.8 – Une illustration de ce à quoi ressemblerait le pavage si $y_1 \neq 0$ et $x_2 \neq 0$.

le couple de coefficients diagonaux d'une matrice sous forme normale de Hermite d'une base de $\mathcal{L}$ (à un ré-ordonnancement près sur les dimensions). Or on sait qu'une telle matrice est unique [12], et qu'il existe deux ré-ordonnancements de deux dimensions (de manière générale, il existe $d!$ ré-ordonnancements de d dimensions, et $2! = 2$). Par conséquent, cela signifie que le couple (m, n) ne peut prendre que deux valeurs différentes au maximum, ce qui nous donne le premier point du théorème.

Pour prouver le second point, nous allons simplement montrer que si R_{x_0,y_0} est une racine primitive de taille $m \times n$, alors les motifs rectangulaires R'_{x_0+1,y_0} et R''_{x_0,y_0+1} de même taille le sont aussi. Le second point sera alors prouvé par récurrence.

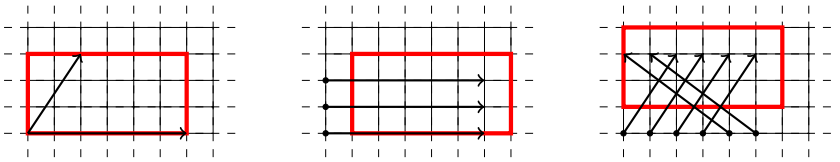
Considérons donc R_{x_0,y_0} une racine primitive de P de taille $m \times n$. Soient également V_1 et V_2 les vecteurs que nous avons défini précédemment. Supposons sans perdre de généralité que $V_1 = (m, 0)$ et $V_2 = (x_2, n)$.

La figure 4.9 montre que R'_{x_0+1,y_0} et R''_{x_0,y_0+1} contiennent les mêmes classes d'équivalence que R_{x_0,y_0}. En effet, dans chacun des cas, il existe une bijection qui préserve les classes d'équivalence entre les pixels du motif original et ceux des motifs translatés. Cette bijection consiste en une translation par un vecteur qui conserve la classe d'équivalence des pixels.

Dans le cas de R'_{x_0+1,y_0}, les vecteurs sont soit V_1, soit $(0, 0)$ (voir figure 4.9b). Dans le cas de R''_{x_0,y_0+1}, les vecteurs sont V_2, $V_2 - V_1$ ou $(0, 0)$ (voir figure 4.9c). Puisque R'_{x_0+1,y_0} and R''_{x_0,y_0+1} contiennent exactement les mêmes classes d'équivalence que R_{x_0,y_0} et sont également des motifs rectangulaires, il s'ensuit qu'ils sont aussi des racines primitives de P. □

Remarquons que la borne supérieure énoncée dans le théorème 4.10 est atteinte. En effet, il existe des images pour lesquelles les racines primitives peuvent avoir deux tailles différentes (en fait, la plupart d'entre elles). Un exemple d'une telle image est donné dans la figure 4.3c. Toutefois, il existe également des images pour lesquelles les racines primitives n'ont qu'une seule taille possible. Nous reviendrons sur ces racines *doubles* un peu plus tard.

Le second point du théorème 4.10 nous permet d'énumérer toutes les racines primitives d'une image donnée. Une telle énumération est présentée dans la figure 4.10.



(a) La racine primitive originale et ses vecteurs de translation V_1 et V_2.

(b) Le nouveau motif R'_{x_0+1,y_0}

(c) Le nouveau motif R''_{x_0,y_0+1}

FIGURE 4.9 – La translation d'une racine primitive préserve ses classes d'équivalence. Ici, $V_1 = (6, 0)$ et $V_2 = (2, 3)$.

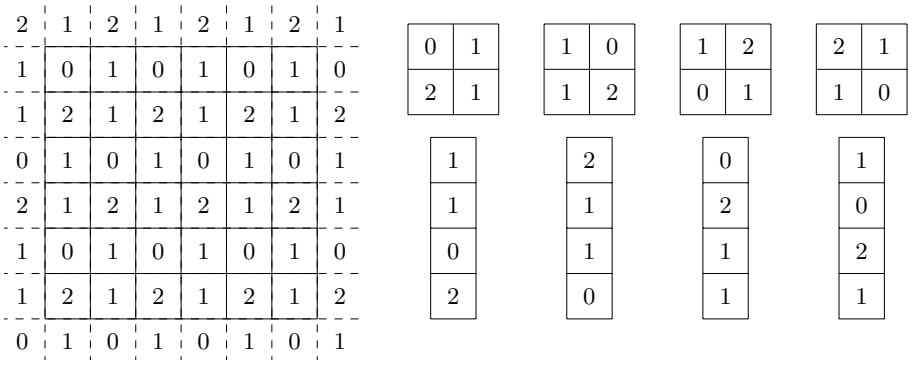


FIGURE 4.10 – L'intégralité des racines primitives de l'image donnée en exemple.

4.1.3 Discussion sur l'ensemble des racines primitives

Nous allons nous intéresser dans cette section à certaines propriétés significatives de la fonction $\mathcal{F}$ qui associe à une image bi-périodique P l'ensemble de ses racines primitives.

« Calculabilité » de la fonction $\mathcal{F}$

Le premier résultat, probablement le plus intéressant, est que la fonction $\mathcal{F}$ est « calculable ». Même si elle s'applique à un objet infini (plus précisément, à un objet infini à support fini, dont la taille est inconnue *a priori*). Les configurations périodiques des automates cellulaires sont de tels objets, et c'est bien entendu à l'aide de ce modèle que nous établirons la calculabilité de $\mathcal{F}$ dans la section 2 de ce chapitre, grâce à un algorithme d'élection de leader.

Injectivité dans le cas général

Il est possible de remarquer que deux images étant la translation l'une de l'autre ont exactement le même ensemble de racines primitives, par conséquent $\mathcal{F}$ n'est clairement pas injective. Nous pouvons en revanche nous intéresser à ce qui se produirait si on définissait nos images à translation près, ce qui constituerait un relâchement raisonnable de notre définition.

Il se trouve que, même dans ce cas de figure, la fonction $\mathcal{F}$ reste non-injective. Un contre exemple permettant de prouver ce point est donné par la figure 4.11,

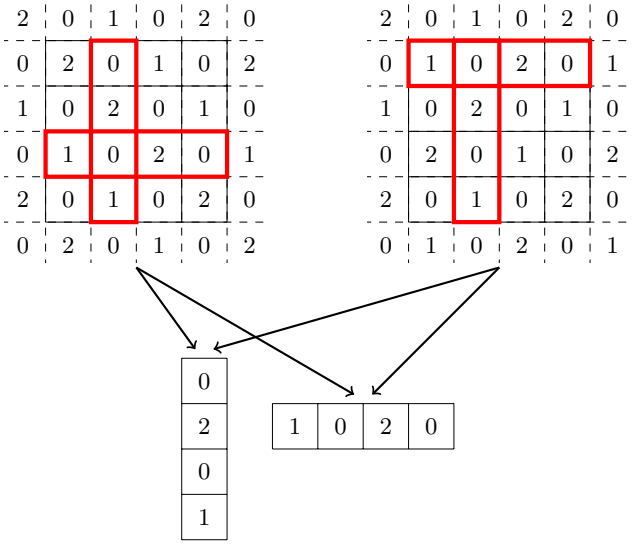


FIGURE 4.11 – La fonction qui extrait les racines primitives n'est pas injective : les deux images ont le même ensemble de racines primitives, qui sont toutes des décalages de celles présentées sur la figure.

qui présente deux images bi-périodiques différentes par translation, qui ont le même ensemble de racines primitives. Nous en déduisons que la simple connaissance de l'ensemble des racines primitives d'une image est insuffisante pour reconstituer cette image. Il est également nécessaire de connaître les vecteurs de répétition associés à un pavage de l'image par ces racines.

À ce point, un lecteur attentif aura peut être remarqué que les deux images présentées sur la figure 4.11 sont des rotations l'une de l'autre, et que l'ensemble de leurs racines primitives est invariant par rotation. Ce lecteur aura pu se demander si l'injectivité de la fonction $\mathcal{F}$ est vraie si les images sont définies à rotation et à translation près. Malheureusement ce n'est pas le cas, il existe des contre-exemples plus complexes qui ne sont pas invariants par rotation.

Le cas particulier d'une racine double

Considérons ce qui se produit lorsque qu'il n'y a qu'une seule dimension possible pour les racines primitives d'une image, c'est à dire lorsque la forme normale de Hermite de la matrice associée à P est diagonale au lieu de simplement triangulaire. Un exemple d'un tel cas est présenté dans la figure 4.12, les racines primitives de P sont alors dites *doubles*.

Si R_{x_0,y_0} est une racine double de taille $m \times n$ d'une image P, cela signifie que les vecteurs de translation associés à R_{x_0,y_0} sont horizontaux et verticaux ($V_1 = (m, 0)$ et $V_2 = (0, n)$). Dans ce cas, P peut aisément être reconstruite en translatant simplement R_{x_0,y_0} selon V_1 et V_2. Ceci implique que $\mathcal{F}$ est bijective sur les images comportant des racines doubles, pour peu qu'elles soient définies à translation près.

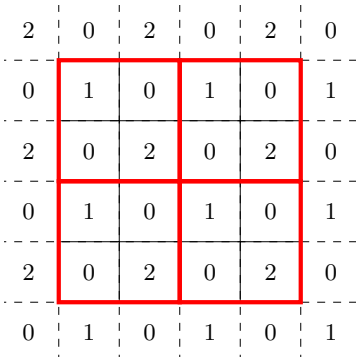


FIGURE 4.12 – Lorsqu'il n'existe qu'une seule dimension pour les racines primitives d'une image, la fonction inverse existe et est triviale.

Propriétés intrinsèques des racines primitives

Il peut être intéressant d'étudier les relations entre les racines primitives que nous venons de définir et les *mots primitifs* de dimension 1 (voir par exemple [41]). En particulier, si P est une image bi-périodique sur Σ et $R = R_{x_0,y_0}$ est une racine de P de taille $m \times n$, alors R peut être considérée comme un mot unidimensionnel horizontal sur l'alphabet Σ^n, ou comme un mot unidimensionnel vertical sur l'alphabet Σ^m (tel que suggéré sur la figure 4.13). Il apparaît qu'au moins un de ces mots unidimensionnels est un mot primitif, tel que défini au chapitre précédent. La preuve de ce point est simple, et réside une fois de plus sur le fait qu'un des vecteurs V_1 ou V_2 a une composante nulle.

En revanche, nous ne savons pas à l'heure actuelle si cette propriété a une réciproque : Étant donnée un motif rectangulaire primitif selon l'une de ses dimensions, la question de savoir s'il existe une image bi-périodique telle que le motif est une racine primitive de cette image est ouverte.

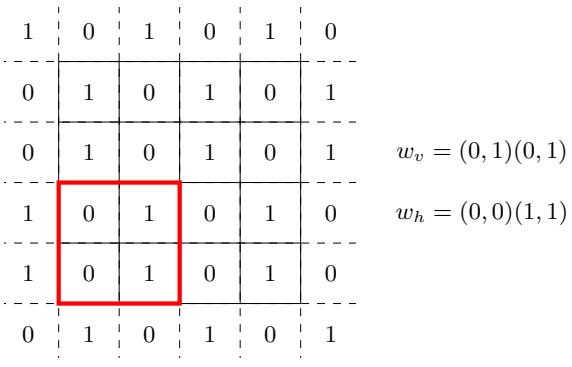


FIGURE 4.13 – Une racine primitive peut être vue comme un mot unidimensionnel vertical w_v sur l'alphabet des tuples horizontaux de Σ, ou comme un mot horizontal w_h sur l'alphabet des tuples verticaux de Σ.

4.2 Algorithme d'élection de leader

Maintenant que nous avons formellement défini les motifs primitifs d'une image bi-périodique, nous allons présenter un algorithme sur automate cellulaire qui va déterminer l'ensemble de ces motifs, par le biais d'une élection de leader. Cet algorithme répond notamment à la question de la calculabilité de l'ensemble des motifs d'une image donnée, que nous nous sommes posée dans la section précédente.

Nous allons travailler dans cette section avec des automates cellulaires *toriques*, tels que définis dans le chapitre 2. Nous rappelons que les automates cellulaires toriques ont $\mathbb{Z}^2$ pour réseau sous-jacent, et qu'ils travaillent sur des configurations bi-périodiques.

4.2.1 Élection de leader sur les automates cellulaires toriques

La problématique de l'élection de leader sur les automates toriques présente les mêmes difficultés qu'en dimension 1 : les configurations sur lesquelles agissent ces automates sont bi-périodiques. Nous pouvons établir entre les cellules de nos configurations une relation d'équivalence définie de la même manière que la relation d'équivalence entre les pixels des images bi-périodiques présentée dans la première section de ce chapitre.

La propriété d'uniformité de la règle de transition des automates cellulaires nous permet d'établir le fait suivant sur les classes d'équivalence des cellules de la configuration initiale :

Fait 4.11. *À tout instant du calcul d'un automate cellulaire torique, toutes les cellules d'une classe d'équivalence donnée sont dans le même état.*

0	1	0	1	0	1
1	0	1	0	1	0
0	0	0	0	0	0
1	0	1	0	1	0
0	1	0	1	0	1
0	0	0	0	0	0

FIGURE 4.14 – Toutes les cellules grisées auront le même comportement durant le calcul, quel que soit l'algorithme utilisé, étant donné qu'elles appartiennent à la même classe d'équivalence.

Définition adaptée de l'élection de leader

Un corollaire des affirmations précédentes est qu'il est impossible d'élire une unique cellule d'une configuration bi-périodique comme leader (il est même

d'ailleurs impossible d'élire un ensemble fini de cellules, quel qu'il soit).

Le problème de l'élection de leader sur les configurations bi-périodiques se résume en fait à l'*élection d'une unique classe d'équivalence*. Nous souhaitons disposer d'un paramètre pour discuter de la complexité en temps des algorithmes sur les automates cellulaires toriques. C'est le but de la prochaine définition.

Définition 4.12 (taille d'une configuration)**.** *La* taille *d'une configuration bi-périodique $\mathcal{C}$ est le nombre de classes d'équivalence des pixels de $\mathcal{C}$. Cette taille est notée N .*

Il nous apparaît que N est le paramètre le plus pertinent lorsqu'on souhaite discuter de la complexité temporelle des algorithmes sur les automates cellulaires toriques.

Résultat principal

Dans cette section, nous allons présenter un algorithme qui effectue l'élection de leader sur les automates cellulaires toriques. Plus précisément, cet algorithme va agir sur les configurations bi-périodiques de telle sorte qu'à partir d'un certain temps $T(N)$:

— Les états des cellules de la classe d'équivalence élue appartiendront à un certain sous-ensemble d'états finaux $F \subseteq \mathcal{Q}$ et ne le quitteront plus.
— Les états de toutes les autres cellules resteront dans $\mathcal{Q} \setminus F$.

Théorème 4.13 (B. [3])**.** *L'algorithme présenté dans cette section résout le problème de l'élection de leader bi-périodique en un temps $T(N)$ polynomial en le nombre N de classes d'équivalences de pixels.*

Au début de l'algorithme, toutes les cellules seront candidates à l'élection. Par la suite, l'algorithme va les éliminer au fur et à mesure, jusqu'à ce qu'il ne reste plus que les cellules d'une seule classe d'équivalence. Notons que de par la nature de notre modèle de calcul, il est impossible pour une cellule unique de s'assurer que le calcul est bel et bien terminé. La terminaison du calcul ne peut être effectivement *observée* que par un agent extérieur au modèle.

L'idée générale de l'algorithme est de regrouper les cellules en blocs contigus, que nous appellerons des *patchs*. De la même manière que les intervalles avaient chacun leur cellule particulière en dimension 1 (la cellule de gauche), chaque patch va élire une de ses cellules comme candidate à l'élection de leader. Au fur et à mesure que ces patchs fusionneront entre eux, l'ensemble de cellules candidates va diminuer jusqu'à ce qu'il ne forme plus qu'une seule classe d'équivalence.

4.2.2 Objets et outils de base

L'objet principal que nous allons considérer est un *patch*. Informellement, un patch est un ensemble fini de cellules adjacentes, dans lequel des calculs seront effectués. Il s'agit du pendant des *intervalles* en dimension 2. Plus précisément, nous cherchons à partitionner toute la configuration en patchs, et à faire en sorte que le comportement d'un patch autour du calcul dépende uniquement de son contenu et du contenu de ses voisins directs, à l'instar de ce qui a été fait en dimension 1.

Plus tard dans la section, nous introduirons des signaux qui voyageront sur la frontière des patchs. Nous voulons que cette frontière puisse être parcourue par

un signal dans son intégralité, sans « saut », ce qui semble interdire qu'un patch ait un trou. Toutefois, notre définition formelle d'un patch nous permettra, dans une certaine mesure, de passer outre cette limitation.

Patchs, frontières et contenu

Pour une définition propre des patchs, nous allons devoir diviser le réseau de cellules. Chaque cellule du réseau va être divisée en quatre sous-cellules (voir figure 4.15). Cette division va entre autres nous servir à considérer les patchs ayant des trous. Nous définirons d'abord les patchs comme des courbes fermées simples, puis comme des ensembles de cellules.

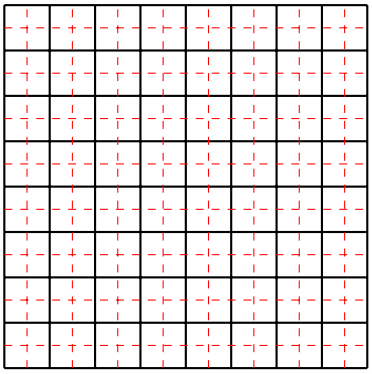


FIGURE 4.15 – Division du réseau de cellules

Définition 4.14 (patchs en tant que courbes)**.** *Un* patch *est une courbe finie, fermée et simple du réseau de cellules divisé, qui obéit à certaines restrictions, qui seront détaillées ci-après.*

Les restrictions portent sur les angles de la courbe, et consistent en un ensemble d'angles autorisés pour la courbe, qui sont explicités sur la figure 4.16. Ces restrictions peuvent être résumées comme suit :

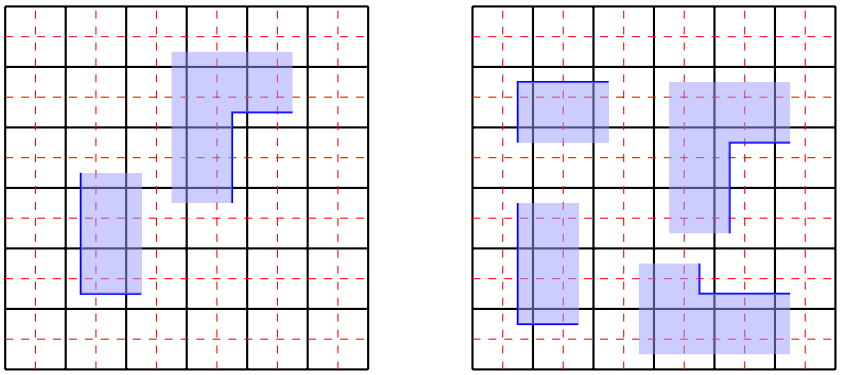
- Chaque angle externe (*i.e.* un angle de 90° mesuré de l'intérieur de la courbe) doit se situer entre l'intérieur d'un coin en ligne pleine et l'extérieur d'un coin en ligne pointillée.
- Chaque angle interne (*i.e.* un angle de 270° mesuré de l'intérieur de la courbe) doit se situer entre l'extérieur d'un coin en ligne pleine et l'intérieur d'un coin en ligne pointillée.

La figure 4.17 donne des exemples de courbes simples du réseau divisé. Notons que les seules courbes définissant des patchs sont celles des figures 4.17a et 4.17b, car celle de la figure 4.17c comporte des angles interdits.

L'intuition derrière ces restrictions est que les patchs ne doivent pas contenir de « demi-cellules », comme c'est le cas sur la figure 4.17c. On doit clairement pouvoir dire si une cellule est « à l'intérieur » ou « à l'extérieur » d'un patch.

Définition 4.15 (frontière)**.** *La* frontière *d'un patch est la projection de sa courbe sur les arêtes extérieures des cellules du réseau originel (non divisé).*

Définition 4.16 (patchs en tant qu'ensemble de cellules)**.** *Le* contenu *d'un patch est l'ensemble des cellules à l'intérieur de sa frontière.*



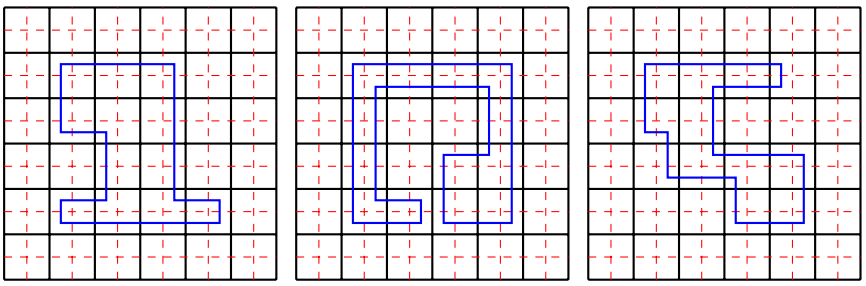
(a) Angles autorisés dans un patch (b) Angles interdits dans un patch

FIGURE 4.16 – Reconnaissance locale des courbes définissant les patchs (la zone colorée dénote l'intérieur de la courbe.)

Lorsque l'abus ne prête pas à confusion, le terme de *patch* pourra désigner son contenu, ou encore les états initiaux des cellules de ce contenu.

Définition 4.17 (taille d'un patch)**.** *La* taille *d'un patch est le nombre de cellules à l'intérieur de sa frontière. Cette taille (aire) est notée a.*

Des exemples de patchs, de leur contenu et de leur frontière sont présentés dans la figure 4.18. Remarquons que notre définition nous permet de considérer des patchs dont le contenu présente un trou, mais dont la frontière peut être parcourue par un signal sans « saut » (voir figure 4.18b).



(a) Patch acceptable (b) Patch acceptable (c) Patch interdit

FIGURE 4.17 – Exemples de courbes simples pouvant définir des patchs.

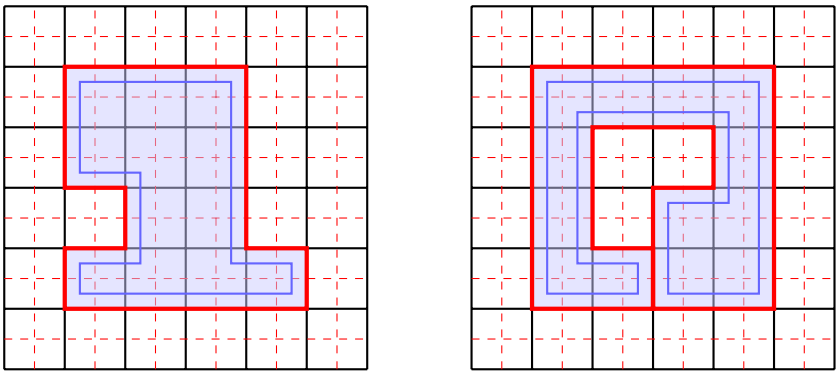
Patchs propres Nous remarquons que certaines parties de la courbe définissant un patch peuvent être sans influence sur son contenu (voir l'exemple de la figure 4.19). La définition suivante nous donne un représentant canonique pour le contenu d'un patch.

Définition 4.18 (patch propre)**.** *Un patch est dit* propre *si la courbe qui le définit ne contient pas le motif présenté dans la figure 4.19a. Le patch propre associé à un patch « brut » est le patch dans lequel toutes les occurrences du*

motif interdit ont été localement supprimées. Remarquons que cette opération de suppression ne modifie pas le contenu du patch.

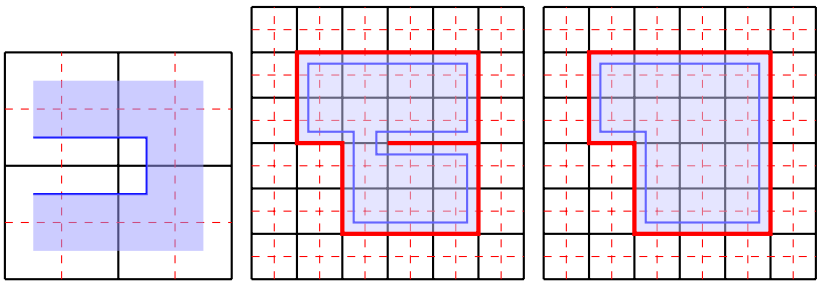
Nous aimerions pouvoir identifier une cellule particulière dans les patchs ; c'est le sens de la définition suivante.

Définition 4.19 (cellule principale d'un patch)**.** *La* cellule principale *d'un patch est la cellule la plus à gauche parmi ses cellules les plus en haut.*



(a) Sans trou (b) Avec un trou

FIGURE 4.18 – Exemples de patchs, avec leurs frontières et leurs contenus.



(a) Le motif interdit (b) Un patch non-propre (c) Un patch propre

FIGURE 4.19 – Transformation d'un patch quelconque en patch propre.

Mot de patch

L'algorithme que nous allons présenter va devoir comparer des patchs entre eux. Un patch étant un objet complexe, nous allons introduire une représentation canonique que notre modèle pourra manipuler. Un patch va être représenté par un mot encodant à la fois la forme de sa frontière et son contenu.

Définition 4.20 (mot de contour)**.** *Le* mot de contour *associé à un patch P est le mot sur l'alphabet $\Lambda = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ qui représente la forme de sa frontière en ordre horaire, en commençant par le coin en haut à gauche de sa cellule principale.*

Par exemple, le mot de contour du patch présenté dans la figure 4.20 est

$$\rightarrow\rightarrow\rightarrow\downarrow\downarrow\downarrow\rightarrow\downarrow\leftarrow\leftarrow\leftarrow\leftarrow\uparrow\rightarrow\uparrow\leftarrow\uparrow\uparrow$$

Nous devons maintenant compléter le mot de contour pour prendre en compte le contenu du patch. Une première solution triviale serait d'énumérer le contenu de toutes ses cellules en ordre de lecture, en commençant par la cellule principale. Toutefois, cette solution présente l'inconvénient de ne pas être calculable localement. À la place, nous nous proposons de construire un *arbre couvrant* sur les cellules d'un patch, dont la racine serait la cellule principale.

Nous admettons pour le moment que la construction d'un tel arbre est possible, une fois que la cellule principale est identifiée, et que cet arbre est unique pour un patch donné (la preuve de ce point sera formulée dans la section 4.2.4). Les arbres couvrants associés à différentes formes de patchs sont présentés sur la figure 4.21.

Définition 4.21 (mot de patch)**.** *Soit Σ l'alphabet d'entrée de notre automate cellulaire le* mot de patch *associé à un* patch propre P *est le mot $w_P = w_\Lambda w_\Sigma$, où $w_\Lambda \in \Lambda^*$ est le mot de contour du patch, et $w_\Sigma \in \Sigma^*$ est le mot représentant le contenu des cellules du patch, ordonné par l'ordre préfixe d'un parcours en profondeur de l'arbre couvrant associé au patch (pour un ordre donné sur les directions).*

Par exemple, voici le mot de patch correspondant au patch propre présenté sur la figure 4.20, dont l'arbre couvrant est celui présenté sur la figure 4.21a :

$$w_P = \rightarrow\rightarrow\rightarrow\downarrow\downarrow\downarrow\rightarrow\downarrow\leftarrow\leftarrow\leftarrow\leftarrow\uparrow\rightarrow\uparrow\leftarrow\uparrow\uparrow\ 011010011010$$

Remarquons qu'il existe une bijection entre un patch propre et son mot de patch (c'est-à-dire qu'il est possible de reconstruire un patch à partir de son mot de patch).

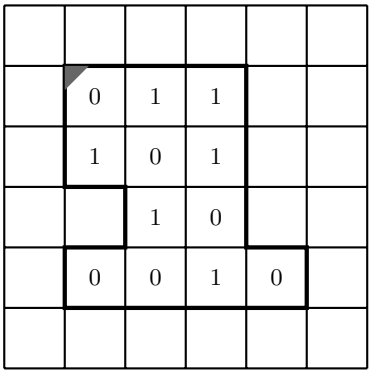


FIGURE 4.20 – Un patch propre et son contenu, avec sa cellule principale mise en évidence.

Voisinages locaux

Nous allons maintenant introduire une partition de la frontière d'un patch propre en *voisinages locaux* de la manière suivante :

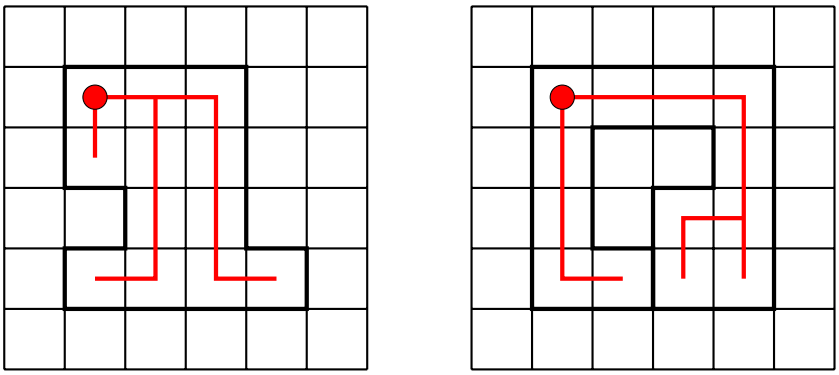


FIGURE 4.21 – Des exemples d'arbres couvrants associés à des patchs, et leurs racines.

Définition 4.22 (voisinage local, voisin local)**.** *Un* voisinage local *d'un patch est une composante connexe maximale de la frontière de ce patch pour laquelle le patch adjacent est unique. Un* voisin local *est alors défini canoniquement comme le patch adjacent à un voisinage local.*

Nous avons prétendu quelques lignes plus haut que cette définition correspondait à une partition de la frontière, ce qui implique que tout patch doit être à tout instant entouré d'autres patchs. Cette condition est vérifiée durant l'exécution de notre algorithme, car la configuration entière est divisée en patchs.

Nous remarquons sur la figure 4.22 que le nombre de voisins locaux d'un patch donné peut être supérieur au nombre de patchs distincts qui lui sont adjacents. De plus, nous remarquons aussi qu'un patch peut être son propre voisin local.

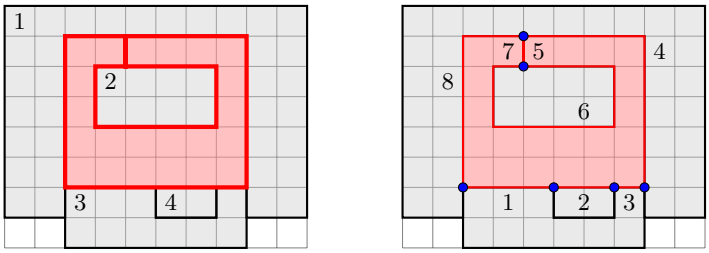


FIGURE 4.22 – Ce patch a 4 patchs qui lui sont adjacents, mais 8 voisins locaux correspondant aux 8 voisinages locaux de sa frontière.

4.2.3 Vue d'ensemble de l'algorithme

Tout au long de cet algorithme, la configuration de l'automate cellulaire va entièrement être partitionnée en patchs, tels que définis précédemment. Au lieu de parler directement du comportement individuel des cellules, et pour une meilleure compréhension de l'algorithme, nous allons tout d'abord considérer

que les patchs sont munis de *méta-états*, et nous allons étudier la façon dont ces méta-états vont évoluer selon un *méta-algorithme*.

À l'initialisation de l'algorithme, la configuration sera entièrement décomposée en patchs de taille 1, c'est à dire que chaque cellule sera seule dans son propre patch. Notons que cette opération peut être effectuée de manière locale. Le but de l'algorithme est de fusionner les patchs les uns avec les autres, jusqu'à ce que la configuration ne soit plus composée que de la répétition périodique du même grand patch. La figure 4.23 présente l'évolution d'une configuration depuis l'initialisation jusqu'à l'état final.

Méta-états et méta-algorithme

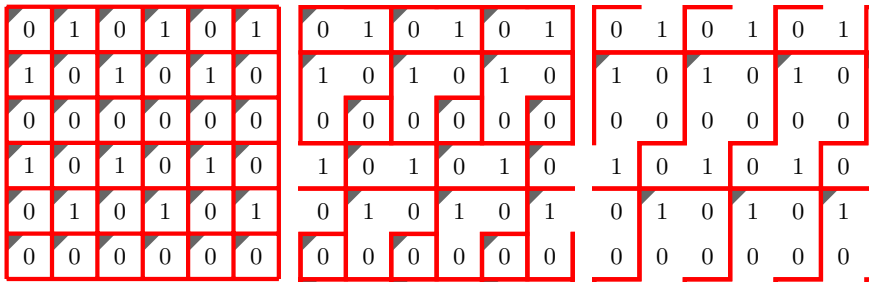
Chaque patch propre aura un méta-état pour chacun de ses voisins locaux (ces méta-états seront appelés états de *fusion*), plus un méta-état spécial (l'état d'*attente*). Les états de fusion peuvent être interprétés comme si le patch essayait de fusionner avec le voisin local associé à l'état, tandis que l'état d'attente signifie que le patch est en train d'effectuer un calcul de plus bas niveau.

Le comportement d'un patch propre va suivre les règles suivantes :

— Au début de son existence, un patch restera dans l'état d'attente pendant un temps fini.

— Après que ce temps initial soit écoulé, son état va effectuer un cycle sur l'ensemble de ses états de fusion, jusqu'à ce que le patch fusionne effectivement.

Nous désirons qu'une fusion se déclenche lorsque une paire de patchs adjacents se trouve simultanément dans un état de fusion correspondant au même voisinage local (c'est à dire lorsque chacun essaie de fusionner avec l'autre, voir figure 4.24). Le processus de fusion effectif et le temps que passeront les patchs dans chaque état de fusion seront détaillés dans les sections suivantes.



(a) État initial (b) État intermédiaire (c) État final

FIGURE 4.23 – Une évolution possible pour une configuration bi-périodique au cours du temps. Les coins grisés représentent les cellules principales des patchs.

Reformulation du résultat principal

Nous allons maintenant expliciter le lien entre les patchs et l'élection de leader. En préambule, il est clair qu'au cours de l'algorithme, un patch donné ne peut pas contenir deux représentants d'une même classe d'équivalence. En effet, deux cellules d'une même classe d'équivalence ont exactement le même

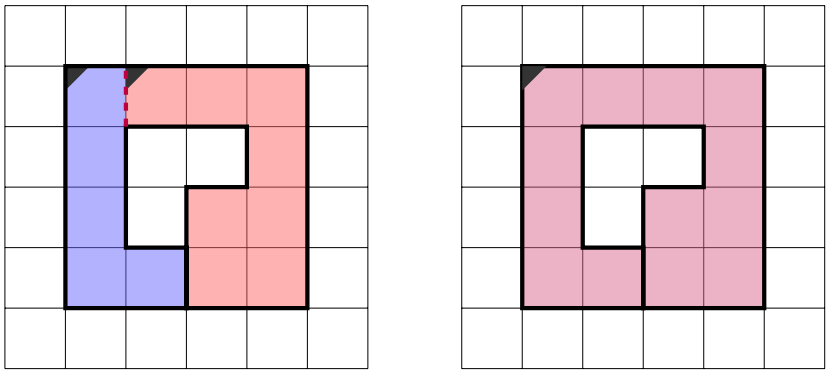


FIGURE 4.24 – Deux patchs fusionnant selon un voisinage local commun.

comportement durant le calcul ; le fait d'avoir ces deux cellules à différentes positions dans un même patch briserait cette symétrie.

Il s'ensuit que la taille maximale que peut atteindre un patch est N, le nombre de classes d'équivalence de la configuration initiale. De plus, si l'algorithme parvient à construire un patch de taille N par fusions successives, alors à ce point du temps la configuration doit être entièrement pavée par des translations de ce patch. Une unique classe d'équivalence peut alors être élue en sélectionnant la cellule principale de chacun des patchs de ce pavage (ces cellules appartiennent évidemment à la même classe d'équivalence). Pour correspondre au formalisme que nous avons utilisé dans le théorème 4.13, nous considérerons qu'une cellule est dans le sous-ensemble final $F \in \mathcal{Q}$ si elle est actuellement la cellule principale d'un patch.

Le but de l'algorithme est donc de fusionner des patchs tant qu'il existe deux patchs adjacents qui sont différents. En effet, si une telle situation se produit, cela signifie que les patchs n'ont pas atteint leur taille maximale. En gardant cela à l'esprit, il suffit de prouver le lemme suivant pour prouver le théorème 4.13 :

Lemme 4.23. *Si à un instant donné il existe deux patchs adjacents dont les mots de patch sont différents, alors au moins l'un d'entre eux doit avoir fusionné en un temps polynomial en leur taille.*

4.2.4 Algorithme détaillé

Préliminaires

Comme nous l'avons annoncé précédemment, chaque cellule de la configuration va être divisée en quatre sous-cellules dans lesquelles se feront les calculs effectifs. Nous parlerons de la *configuration divisée* quand il sera nécessaire de considérer les quatre sous-cellules de chaque cellule, et de la *configuration unifiée* dans le cas contraire. L'ensemble des états de l'automate est donc $\mathcal{Q} = \Sigma \times \mathcal{Q}_s^4$, où Σ est l'alphabet d'entrée de l'automate, et $\mathcal{Q}_s$ est l'ensemble des états des sous-cellules. Nous voulons que l'information d'entrée soit accessible à tout instant dans une cellule, par conséquent la projection de $\mathcal{Q}$ sur Σ reste invariante au cours du temps.

Si chaque sous-cellule est associée à un coin de la cellule originale, nous obtenons la division informelle qui a été utilisée pour définir les patchs.

Pour rendre possible la simulation des méta-états de fusion, chaque patch va construire et maintenir un signal qui va parcourir sa frontière. Plus précisément, ce signal va effectuer un cycle en suivant les arêtes de la frontière, en s'arrêtant parfois plus d'une unité de temps sur une arête. Nous allons maintenant définir les points sur lesquels va voyager le signal.

Définition 4.24 (points d'attente et périmètre d'un patch)**.** *Les* points d'attente *d'un patch sont les intersections de la courbe qui le définit avec les lignes de division des cellules (voir figure 4.25). Le* périmètre *d'un patch est le nombre de ses points d'attente. Ce périmètre est noté n.*

Remarquons qu'il existe au plus quatre points d'attente dans une unique cellule ; par conséquent chacun d'entre eux peut être associé à une sous-cellule. De plus, on peut faire correspondre chaque point d'attente à une unique cellule voisine de la cellule dans laquelle il se trouve, et par conséquent à un unique voisinage local du patch. Pour faire le lien avec le méta-algorithme, nous considérons qu'à un instant donné, un patch est dans le méta-état de fusion associé au point d'attente sur lequel son signal se trouve. Un patch sera dans le méta-état d'attente tant qu'il ne contiendra pas encore de signal de fusion.

Puisqu'on utilise le voisinage de Moore sur la configuration unifiée, nous remarquons que le signal peut se déplacer d'un point d'attente à un autre en une unité de temps.

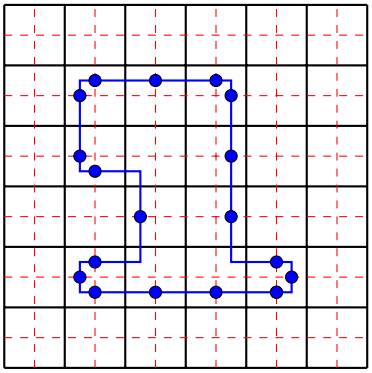


FIGURE 4.25 – Mise en évidence des points d'attente d'un patch de taille 12 et de périmètre 18.

Le signal va encoder le mot associé au patch sous la forme de temps d'attente sur la frontière. Il va se comporter de telle sorte que les signaux de deux patchs différents vont finir par se désynchroniser et se rencontrer, provoquant ainsi une fusion.

Étape initiale

À l'instant initial de l'algorithme, l'automate va construire des patchs de taille 1 sur tout la configuration. Le résultat de cette opération, qui peut être effectuée localement, est visible sur la figure 4.26. Nous faisons en sorte que les patchs retiennent leur information d'entrée à tout instant.

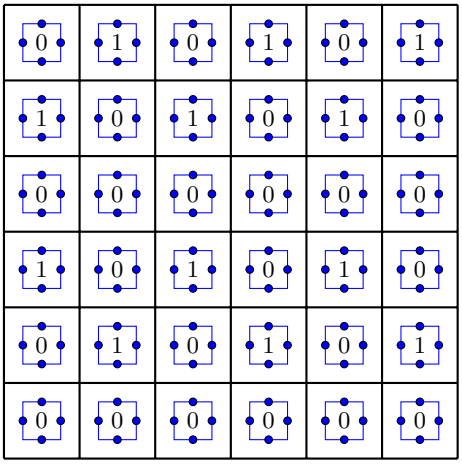


FIGURE 4.26 – Les patchs issus de la configuration initiale. Les lignes de division des cellules ont été masquées par souci de clarté.

Comportement des patchs au cours du temps

Dans cette section, nous allons considérer le cycle de vie d'un patch, depuis sa création jusqu'au moment de sa « mort » (en fait, de sa fusion). Nous supposons que la frontière d'un patch est *synchronisée*, comme le ferait un algorithme de ligne de fusiliers (voir le chapitre 2, [5] ou [52]). Cette supposition est vérifiée au début du calcul, puisque ce début est en lui-même un événement synchronisant. nous verrons plus tard comment un patch peut se synchroniser lorsqu'il vient d'être créé. Nous ne faisons pas la supposition qu'un patch est *propre* au début de son existence.

La vie d'un patch va se dérouler en 5 étapes :

— sélection de la cellule principale ;
— réparation du patch en patch propre ;
— construction de l'arbre couvrant ;
— lancement du signal ;
— fusion et synchronisation du nouveau patch.

Sélection de la cellule principale

Considérons un patch nouvellement créé, que nous supposons synchronisé. La première tâche qui va être accomplie est l'identification de la cellule principale, telle que définie précédemment. Considérons l'ensemble des sous-cellules qui forment la courbe fermée servant à définir le patch. Ces sous-cellules peuvent être identifiées localement, et forment une structure d'anneau dans le plan bi-dimensionnel. La cellule principale du patch est la cellule contenant la sous-cellule la plus à gauche parmi celles qui sont le plus en haut (voir figure 4.27).

Il se trouve que l'élection d'une cellule particulière sur une structure d'anneau par un automate cellulaire est un sujet bien connu et étudié, et qu'il existe un article de Nichitiu, Mazoyer et Rémila [39] qui nous fournit un algorithme qui répond exactement à nos besoins. Nous avons supposé que la frontière de l'automate était synchronisée, tout ce qu'il nous reste à faire est donc de simuler l'algorithme présenté dans [39] dans le sous-automate (c'est à dire l'automate

dont les cellules sont les sous-cellules de notre configuration). Ensuite, il nous suffira de sélectionner comme cellule principale la cellule dont la sous-cellule en haut à gauche aura été élue par l'algorithme. Il convient de noter que cet algorithme s'exécute en un temps polynomial en la taille du patch.

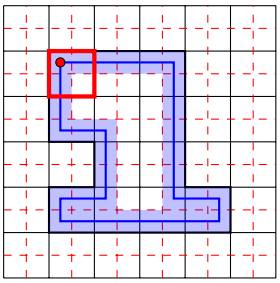
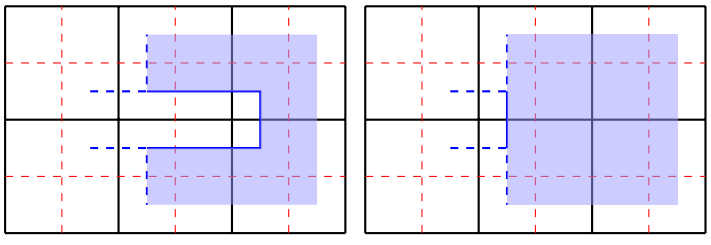


FIGURE 4.27 – Mise en évidence de la structure d'anneau de la frontière.

Réparer le patch

Nous voulons maintenant transformer notre patch brut en patch propre. Cette opération peut être effectuée localement en supprimant toutes les occurrences du motif interdit dans un patch propre (voir figure 4.19a). D'un point de vue pratique, un signal va être envoyé depuis la cellule principale le long de la frontière. Ce signal va remplacer les motifs selon la règle présentée dans la figure 4.28. On laisse au lecteur le soin de se convaincre que l'application itérative de cette règle suffit à transformer un patch quelconque en un patch propre qui lui est associé (voir figure 4.19).

Notons également que le nombre de motifs interdits à supprimer est au maximum linéaire en la taille du patch, cette étape s'effectue donc également en temps polynomial.



(a) Avant (b) Après

FIGURE 4.28 – La transformation locale qui change un patch quelconque en patch propre. La zone en bleu dénote l'intérieur du patch.

Construction et parcours de l'arbre couvrant

Maintenant que le patch est propre et que la cellule principale est identifiée, nous pouvons construire l'arbre couvrant à partir de la cellule principale par propagation en suivant une règle simple : si à un instant donné une cellule du patch a une voisine à l'intérieur de ce patch dans laquelle l'arbre a été construit,

alors à l'instant suivant une arête de l'arbre va être créée entre la cellule et sa voisine. Si une cellule a plusieurs voisines remplissant ces conditions, nous en choisissons une selon un ordre prédéfini sur les directions (par exemple, haut > gauche > bas > droite).

La figure 4.29 présente la construction d'un arbre couvrant sur un patch d'exemple. Pour des raisons pratiques, l'arbre est orienté vers la racine.

Nous désirons maintenant parcourir l'arbre et fournir à la cellule principale les symboles contenus dans les cellules sur demande. Ceci peut être fait de manière simple en simulant un automate à état fini qui effectuerait un parcours en profondeur de l'arbre et fournirait un symbole à la racine chaque fois qu'il découvrirait une nouvelle cellule. Remarquons qu'une telle méthode nous garantit un intervalle de temps entre la réception de deux symboles consécutifs linéaire en la taille de l'arbre.

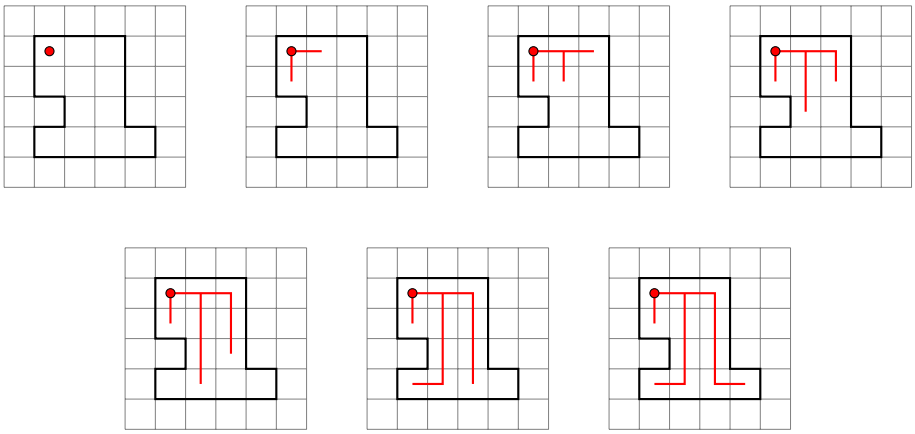


FIGURE 4.29 – Construction d'un arbre couvrant d'un patch à partir de la cellule principale.

Lancement du signal

Dès que le patch est propre, la cellule principale va envoyer le signal qui déclenchera la fusion entre les patchs. Le comportement de ce signal est détaillé dans cette section.

Considérons l'automate cyclique unidimensionnel dont les cellules sont les points d'attente d'un patch. Cet automate est appelé l'*automate frontière*. Sa taille est le périmètre du patch dans lequel il est simulé, qui est notée n. Le lecteur se convaincra que l'automate frontière peut être simulé pas-à-pas par notre automate cellulaire bi-dimensionnel.

Définition 4.25 (point d'attente principal)**.** *Le* point d'attente principal *d'un patch est le point d'attente situé en haut de la cellule principale d'un patch.*

Par définition de la cellule principale, il est évident que ce point existe.

Nous voulons que le point d'attente principal — qui est une cellule de l'automate frontière — ait accès au mot de patch. Ceci peut être accompli en parcourant la frontière du patch et son arbre couvrant, et en envoyant le bon symbole jusqu'à la cellule principale. Le point d'attente principal va stocker un unique

symbole à chaque instant, en commençant par le premier symbole du mot de patch (qui se trouve systématiquement être le symbole « →"), et en envoyant un signal lorsqu'il aura besoin de récupérer le prochain symbole. Nous savons que le temps pour récupérer un symbole est linéairement borné par la taille du patch, ce qui signifie qu'il est borné quadratiquement par le périmètre n du patch. Nous affirmons que ce temps n'introduira pas de délai dans l'algorithme, nous remarquerons en effet que le délai entre deux requêtes de symbole sera en $O(n^3)$. Dans les sections suivantes, le mot du patch sera noté $a = a_0 a_1 \dots a_l$, et i va dénoter la position du symbole a_i contenu dans le point d'attente principal.

Le point d'attente principal va donc envoyer un signal de fusion dans l'automate frontière. Ce signal va parcourir l'automate en sens anti-horaire à vitesse maximale ; il transportera parfois un galet, que nous appellerons le *galet d'attente* (voir figure 4.30). Lorsque le signal de fusion est créé, le galet d'attente est positionné sur le point d'attente principal, qui stocke le symbole a_0. Le signal obéit ensuite aux règles suivantes, qui sont illustrées sur la figure 4.30 :

- Tourner autour de l'automate frontière à vitesse 1 (*i.e.* effectuer un cycle sur la frontière) jusqu'à ce qu'on rencontre le galet d'attente (voir figure 4.30a).
- Lorsque le galet d'attente est rencontré, le déplacer jusqu'au point d'attente suivant, et attendre un certain *temps d'attente* τ_n sur ce point (voir figure 4.30b).
- Si le galet a été déposé sur le point d'attente principal, effectuer un certain nombre de *cycles supplémentaires* après l'attente. Le nombre exact de cycles dépend de a_i. La cellule principale va ensuite récupérer le prochain symbole (a_{i+1}, ou de nouveau a_0 si a_i était le dernier symbole du mot), et le signal reprend son comportement normal (voir figure 4.30c).

La figure 4.32 résume le comportement du signal et du galet sur un automate frontière simple avec $n = 4$. À tout instant de l'existence du signal de fusion, s'il rencontre le signal de fusion d'un autre patch (voir figure 4.31a), alors les deux signaux sont immédiatement supprimés et une fusion se produit. Ce processus sera détaillé dans les paragraphes suivants.

Définition 4.26 (temps d'attente)**.** *Le temps d'attente associé au périmètre n est le temps $\tau_n = k.n^2$ où $k = \|\Sigma \cup \Lambda\| = \|\Sigma\| + 4$ est le nombre de symboles différents dans le mot de patch. Rappelons que Λ est l'alphabet des 4 directions codant la forme du contour du patch.*

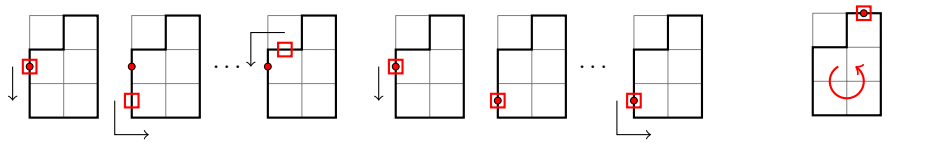
Définition 4.27 (cycles supplémentaires)**.** *Le nombre de cycles supplémentaires que doit effectuer le signal sur l'automate de frontière vaut $val(a_i)$, où val est une bijection quelconque de $\Sigma \cup \Lambda$ dans $[\![0; k-1]\!]$.*

Rappelons que ces cycles supplémentaires se produisent uniquement lorsque le galet est déposé sur le point d'attente principal, ce qui introduit un décalage temporel de $n.val(a_i)$ dans le comportement du signal de fusion.

Notons que la fonction τ_n est constructible en temps quelle que soit la valeur de n. En d'autres termes, il est possible de « compter » jusqu'à τ_n (voir chapitre 2 page 12).

Fusion locale et synchronisation

Lorsque deux signaux de fusion se rencontrent sur la frontière commune à deux patchs, cela signifie que les patchs doivent fusionner, selon la règle globale de

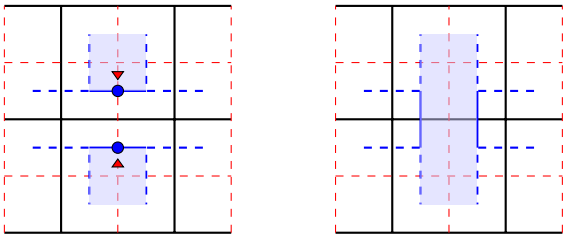


(a) Phase de Déplacement (b) Déplacement du galet et attente (c) Cycles supplémentaires

FIGURE 4.30 – Vue d'ensemble du comportement du signal de fusion dans un patch d'exemple. Le cercle représente le galet d'attente, et le carré représente le signal.

notre méta-algorithme. Cette fusion peut s'effectuer de manière locale, en suivant le processus illustré sur la figure 4.31. Il suffit de supprimer les portions de frontière sur lesquelles se trouvent actuellement les signaux, et de « recoller » les frontières ensemble pour obtenir la frontière d'un nouveau patch, probablement non propre. Remarquons que la courbe qui résulte de cette opération respecte toujours les conditions pour définir un patch.

Une fois que la fusion est effectuée, il est nécessaire d'effacer les deux anciens arbres couvrants, en envoyant des signaux dans le nouveau patch. Ces signaux vont aussi effacer l'information des cellules principales des anciens patchs. Il est important de noter que le processus de fusion s'effectue de façon *non concurrente* entre les patchs : étant donné que chaque patch possède un unique signal de fusion, et que la fusion est déclenchée par la présence en un même point de deux signaux de fusion, il est impossible qu'un même patch soit impliqué dans deux fusions simultanément.



(a) Avant (b) Après

FIGURE 4.31 – Une fusion locale se produit lorsque deux signaux de fusion se rencontrent.

Il faut maintenant synchroniser la frontière du patch issu de la fusion, pour nous retrouver dans la situation décrite dans la section 4.2.4. Nous allons encore une fois considérer la frontière de notre nouveau patch comme un automate cellulaire unidimensionnel cyclique. Il est facile d'identifier une cellule particulière de cet automate, par exemple l'une des cellules sur lesquelles la fusion s'est produite. Nous allons ensuite nous servir de cette cellule comme d'un général pour un algorithme de ligne de fusiliers, dont l'exécution va nous permettre de resynchroniser la frontière, et ainsi permettre la sélection de la cellule principale

sur le nouveau patch. Rappelons nous qu'un algorithme de ligne de fusiliers a été présenté au chapitre 2, page 17. Notons que cet algorithme ou ses variantes s'exécutent en temps linéaire.

« Terminaison » de l'algorithme
Maintenant que le comportement des patchs a été décrit, il nous faut discuter de la fin du calcul. Lorsque les patchs sont de taille maximale (c'est à dire N), ils ne peuvent plus fusionner, de par la définition même des classes d'équivalence : on se souviendra en effet que les patchs maximaux ne peuvent pas fusionner. À la fin de l'algorithme la configuration sera donc entièrement décomposée en patchs identiques (de taille maximale) dans chacun desquels le même signal de fusion continuera sans fin sa course, de façon cyclique.

4.2.5 Preuve de l'algorithme

Dans cette section nous démontrerons que notre construction satisfait le lemme 4.23, c'est à dire que s'il existe à un instant donné deux patchs adjacents dont les mots de patch sont différents, alors au moins un d'entre eux doit avoir fusionné dans un laps de temps polynomial.

Nous supposons qu'il existe deux patchs P_1 et P_2 dont les mots de patch sont différents, et nous allons montrer que si aucun d'entre eux n'a fusionné avec un troisième patch (auquel cas notre propriété serait vérifiée), alors ils vont fusionner ensemble en temps polynomial. Nous noterons n_1 et n_2 les périmètres et i et j la position du symbole sur les mots de patch de P_1 et P_2 respectivement. Nous allons étudier deux cas disjoints : le cas où les périmètres des patchs sont différents, puis le cas où ces périmètres sont égaux.

Preuve du théorème 4.13. **Cas 1 : périmètres différents**

Supposons en premier lieu que $n_1 \neq n_2$. Sans perte de généralité, il est possible d'affirmer que $n_1 < n_2$, ou encore $n_2 \geq n_1 + 1$. Considérons maintenant un instant où le galet d'attente de P_2 est lâché sur un point de la frontière commune avec P_1. Le signal de fusion de P_2 va attendre le temps $\tau_{n_2} = k.{n_2}^2$ sur ce point d'attente. Nous avons donc :

$$\begin{gathered}\tau_{n_2} = k.{n_2}^2 \\ \tau_{n_2} \geq k.(n_1+1)^2 \\ \tau_{n_2} > k.{n_1}^2 + 2k.n_1\end{gathered}$$

Considérons maintenant le temps maximal noté τ_{abs} pendant lequel le signal de fusion de P_1 peut être *absent* du point correspondant sur P_1. Étant donné que le signal de fusion effectue au moins un tour complet de la frontière entre deux attentes, ce temps τ_{abs} est la somme du temps d'attente de P_1, soit τ_{n_1}, et du temps pour le signal de faire un tour complet autour de P_1, qui se trouve être $n_1 - 1$. Nous avons donc :

$$\begin{gathered}\tau_{abs} = \tau_{n_1} + n_1 - 1 \\ \tau_{abs} = k.n_1^2 + n_1 - 1\end{gathered}$$

Il est clair que $\tau_{n_2} > \tau_{abs}$, ce qui signifie que le signal de fusion de P_1 va rencontrer celui de P_2 pendant le temps d'attente de ce dernier. Par conséquent, une fusion va se produire entre P_1 et P_2.

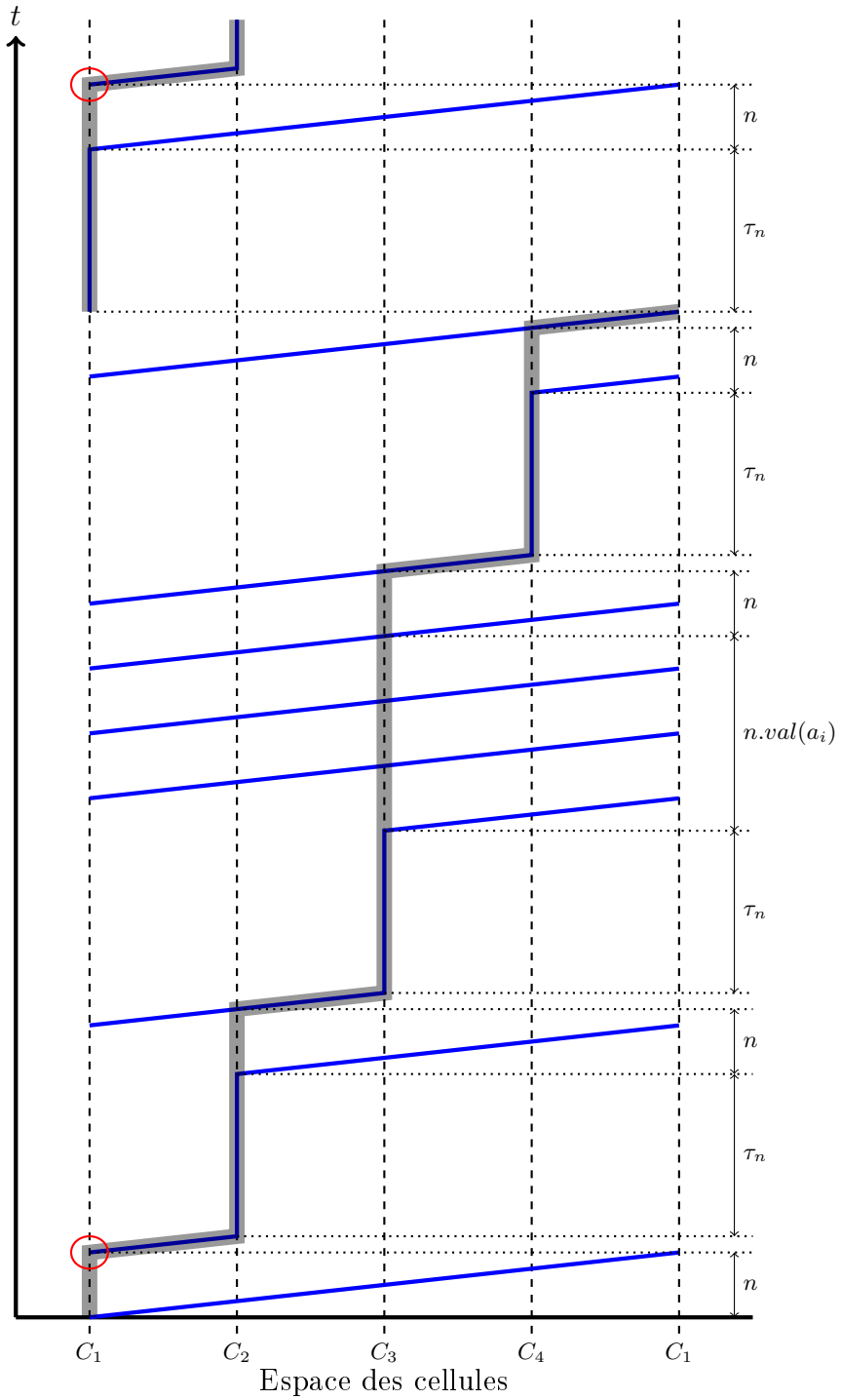


FIGURE 4.32 – Évolution au cours du temps de la position du signal de fusion (ligne fine) et du galet d'attente (ligne épaisse).

Notes concernant la figure 4.32
- Pour ce patch, $n = 4$. Nous supposons que C_3 est le point d'attente principal et que $val(a_i) = 3$ pour l'instant considéré.
- La somme des intervalles de temps non annotés est n, ce qui correspond aux n déplacements du galet jusqu'à sa prochaine position, chacun étant de taille 1.
- L'automate frontière est dans la même configuration aux deux temps encerclés, à la seule différence que le symbole a_i stocké dans le point d'attente principal a pris sa valeur suivante.

Cas 2 : périmètres égaux

Considérons maintenant le cas $n_1 = n_2 = n$. Nommons S_1, S_2, a et b les signaux de fusion et les mots de patch de P_1 et P_2 respectivement. Puisque P_1 et P_2 sont différents, cela signifie que a et b le sont aussi. Nous allons maintenant observer les allées et venues de signaux en un unique point de la frontière commune entre P_1 et P_2.

Considérons le comportement de S_1 sur ce point d'attente : il y passe fréquemment, et s'y arrête parfois pour un temps τ_n. Appelons $(t_k)_{k\geq 0}$ la suite des instants pour lesquels S_1 s'arrête sur le point d'attente pendant un temps τ_n. Par construction de l'algorithme, il y a exactement un instant entre chaque t_k et t_{k+1} pour lequel S_1 attend sur le point d'attente principal de P_1 et effectue $val(a_i)$ cycles supplémentaires autour de la frontière, pour un certain a_i. Le même raisonnement vaut pour S_2, qui attendra sur le point d'attente principal de P_2 et effectuera $val(b_j)$ cycles supplémentaires une unique fois entre chaque t_k et t_{k+1}. Il est donc possible d'associer un couple de symboles (a_{i_k}, b_{j_k}) à chaque t_k.

Nous savons que $a \neq b$, et par construction a ne peut pas être obtenu par permutation circulaire de b (car l'alphabet de contour est distinct de l'alphabet de contenu). Par conséquent, il existe un k_0 pour lequel nous avons $a_{i_{k_0}} \neq b_{j_{k_0}}$. Introduisons quelques moments particuliers, qui sont explicités sur la figure 4.33 :

Notations 4.28 (instants particuliers : T_0, δ)**.**
- *T_0 est le dernier instant où le signal S_2 était présent sur le point d'attente considéré avant l'instant t_{k_0} associé à k_0.*
- $\delta = t_{k_0} - T_0$.

Notons qu'il y a deux types de « trous » pendant lesquels un signal peut être absent d'un point particulier de la frontière : les trous de taille n et les trous de taille $\tau_n + n$. Nous savons que S_1 est présent sur le point d'attente pour une durée τ_n, à partir de l'instant $T_0 + \delta$. Si S_1 rencontre S_2, alors P_1 et P_2 vont fusionner, et la preuve est terminée. Sinon, cela signifie que cette présence de S_1 correspond à un trou de taille $\tau_n + n$ pour S_2, puisqu'il est clair que $\tau_n > n$. Nous savons donc donc que S_2 sera présent au point d'attente au temps $T_0 + \tau_n + n$.

Notons maintenant que le galet d'attente effectue un tour complet autour de l'automate en temps $n.(\tau_n + n + 1) + n.val(a_i)$ sur P_1, et $n.(\tau_n + n + 1) + n.val(b_j)$ sur P_2. nous allons introduire quelques instants supplémentaires, pour la compréhension desquels le lecteur est fortement encouragé à se référer à la figure 4.33 :

Notations 4.29 (instants particuliers : T_1, T_2, δ')**.**

- $T_1 = T_0 + \delta + n.(\tau_n + n) + n.val(a_i) + n$ *est le prochain temps pour lequel* S_1 *va attendre pendant* τ_n *sur le point d'attente. Remarquons que notre formalisme précédent nous donne également* $T_1 = t_{k_0+1}$.
- $T_2 = T_0 + n.(\tau_n + n) + n.val(b_j) + n$ *est un temps pour lequel on est assuré que* S_2 *sera de retour sur le point d'attente, à cause du comportement quasi-périodique de notre signal (voir aussi la figure 4.32). Nous savons aussi que* S_2 *sera présent sur le point d'attente au temps* $T_2 + \tau_n + n$.
- $\delta' = T_1 - T_2$ *est la différence entre ces deux temps.*

Nous allons maintenant prouver que si S_1 et S_2 ne se sont pas rencontrés avant, alors il vont se rencontrer soit à l'instant T_2, soit à l'instant $T_2 + \tau_n + n$.

Nous savons par définition que $0 < \delta < \tau_n + n$. De plus, il est nécessaire que $\delta < n$: le cas contraire indiquerait que S_1 est de retour sur le point d'attente avant que S_2 ne parte. Finalement, nous avons donc $0 < \delta < n$. Il est également clair que $\delta' = \delta + n.(val(a_i) - val(b_j))$. Posons $\Delta = val(a_i) - val(b_j)$. Par construction, nous avons $\Delta \in [\![-k+1, -1]\!] \cup [\![1, k-1]\!]$, car $a_i \neq b_j$. Considérons maintenant deux sous-cas : soit $\Delta < 0$, soit $\Delta > 0$.

Cas 1 : $\Delta < 0$
Supposons dans un premier temps que $\Delta \in [\![-k+1, -1]\!]$. L'inégalité $0 < \delta < n$ nous permet d'affirmer :

$$\begin{gathered} \delta + n.(1-k) \leq \delta' \leq \delta - n \\ n.(1-k) < \delta' < 0 \end{gathered}$$

Cet intervalle de valeurs pour δ' nous assure que les deux signaux vont se rencontrer à l'instant T_2.

Cas 2 : $\Delta > 0$
Supposons maintenant que $\Delta \in [\![1, k-1]\!]$. L'inégalité $0 < \delta < n$ nous permet d'affirmer :

$$\begin{gathered} \delta + n \leq \delta' \leq \delta + n.(k-1) \\ n < \delta' < k.n \end{gathered}$$

Cet intervalle de valeurs pour δ' nous assure que les deux signaux vont se rencontrer à l'instant $T_2 + \tau_n + n$ (ce qui se produit sur la figure 4.33).

□

4.2.6 Analyse temporelle

Souvenons nous que N est le nombre de classes d'équivalence de la configuration initiale, et qu'il s'agit d'une borne supérieure pour la taille de tout patch durant les calculs. Étant donné que le périmètre n d'un patch est dans le pire des cas linéaire en sa taille nous savons donc que pour tout patch $n \in O(N)$.

Le point clé de la section précédente était de montrer que si deux patchs différents sont adjacents, alors au moins l'un d'entre eux doit fusionner avant un certain temps. Nous savons que cette fusion doit se produire avant que les deux patchs n'aient effectué un cycle complet sur leur mot de patch respectif. Souvenons nous que le point d'attente principal change le symbole qu'il contient

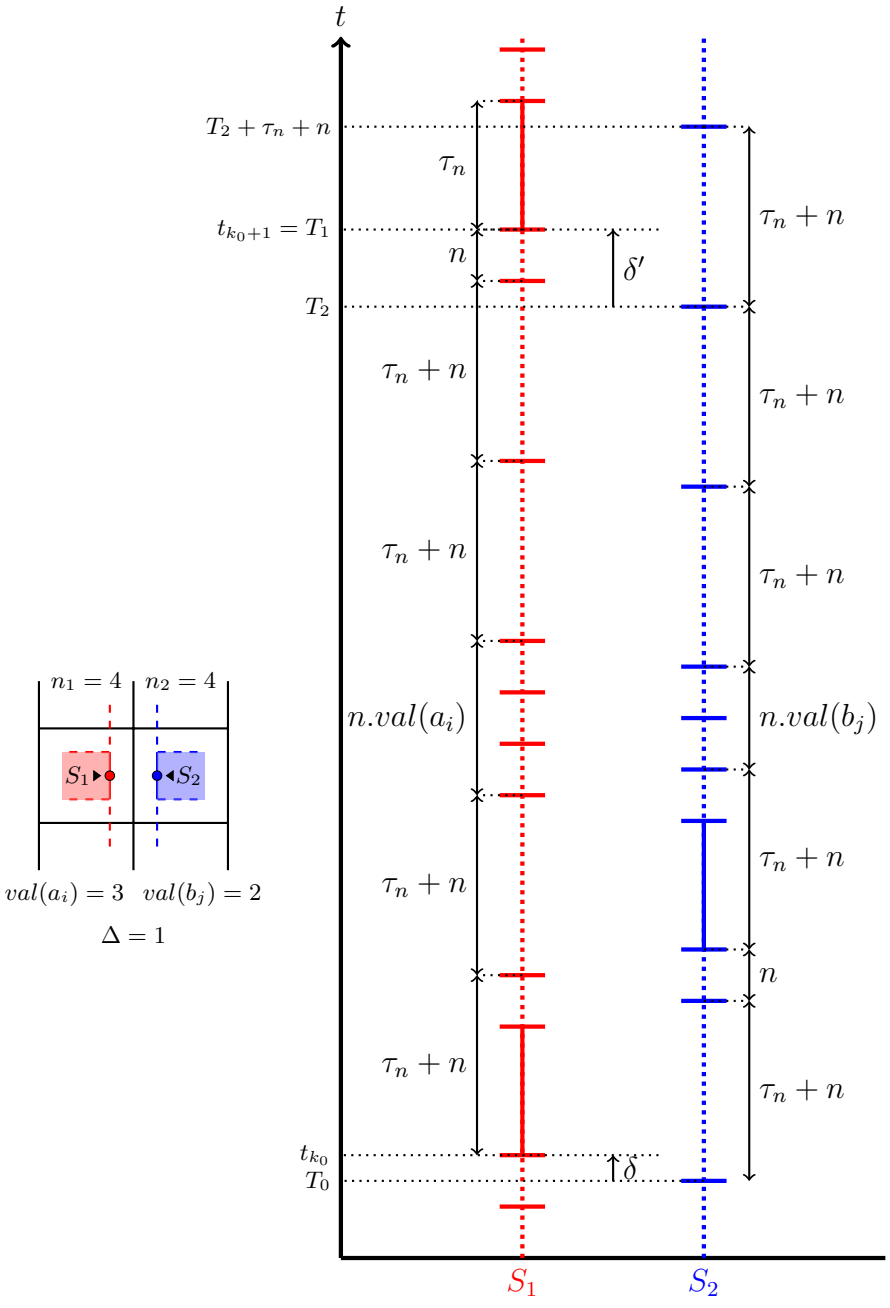


FIGURE 4.33 – Le diagramme de présence des signaux de fusion sur un point d'attente commun à P_1 et P_2. La ligne pleine indique que le signal est présent, le ligne pointillée indique qu'il est absent. Remarquons que le diagramme de présence de S_1 représente exactement la présence du signal sur le cellule C_2 de la figure 4.32. Similairement, le diagramme de S_2 pourrait être celui d'une cellule voisine, pour laquelle la valeur du symbole stocké dans le point d'attente principal serait $val(b_j) = 2$.

tous les $O(n^3)$ unités de temps. Étant donné que la taille du mot de patch est en $O(N)$, nous sommes assurés qu'au moins un des deux patchs considérés va fusionner en temps $O(N \times n^3) = O(N^4)$. Remarquons par ailleurs que le coût des opérations nécessaires à la fusion propre des patchs est bien inférieur à $O(N^4)$.

Finalement, étant donné qu'il existe N patchs distinguables au début des calculs (un par classe d'équivalence de cellules), et qu'au moins deux d'entre eux fusionnent tous les $O(N^4)$ unités de temps, nous sommes assurés d'élire une unique classe d'équivalence de cellules en temps $O(N^5)$.

Cette analyse est extrêmement grossière, elle ne prend pas en compte la nature parallèle du modèle de calcul, et ne considère que deux patchs à un instant donné. Nous sommes convaincus qu'une analyse plus précise établirait un temps $O(N^4)$.

4.2.7 Remarques finales sur l'algorithme

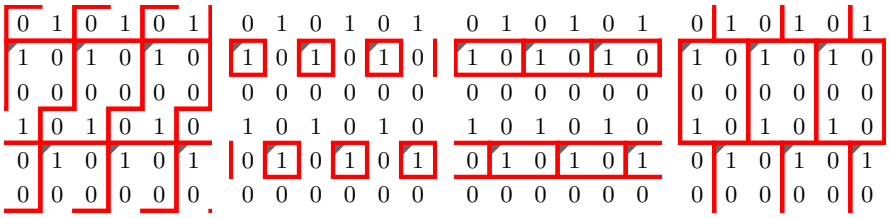
Le lecteur attentif aura remarqué que le calcul ne se *termine* pas dans les faits, il boucle simplement sur un ensemble fini de configurations. Pour parler informellement, cette terminaison est similaire à la *reconnaissance faible* dont nous avons parlé pour les automates de dimension 1. Une version plus forte serait de faire en sorte que la configuration atteigne un point fixe, plutôt qu'un cycle limite. Quelques modifications mineures permettraient à notre algorithme d'atteindre cette propriété. En effet, si un patch a attendu un temps $O(n^4)$ en se rendant compte que ses voisins n'ont pas fusionné, il est capable de déterminer que ses voisins sont des copies de lui-même, et peut geler ses calculs. Il les reprendra éventuellement plus tard, à l'occasion de la fusion de l'un de ses voisins. Si tous les patchs ont déterminé de cette manière que leurs voisins leur sont identiques et se sont gelés, alors la configuration atteint un point fixe.

Lien avec les motifs primitifs

Remarquons que notre algorithme n'effectue pas simplement une élection de leader, il calcule un polyomino qui pave la configuration par translation (ce polyomino est le patch terminal). Nous savons, grâce à [7], que ce polyomino est de forme pseudo-hexagonale. En fait, il s'agit même d'un pseudo-rectangle : nos patchs terminaux peuvent être déformés pour les transformer en rectangles (voir figure 4.34).

Pour ce faire, il suffit de considérer des rectangles contenant uniquement les cellules principales des patchs (figure 4.34b), de les étendre à droite jusqu'à rencontrer une autre cellule principale (figure 4.34c), puis de les étendre en bas jusqu'à ce qu'à rencontrer une troisième cellule principale (figure 4.34d). Chaque rectangle ainsi créé contient exactement un représentant de chaque classe d'équivalence.

Il se trouve que cette méthode de transformation des patchs terminaux en rectangles nous permet d'obtenir les *motifs primitifs* de la configuration initiale (illustrés ici par la figure 4.34d). Plus exactement, nous obtenons l'une des deux formes possibles pour les motifs primitifs. Si nous désirions obtenir la seconde forme, il faudrait d'abord étendre les rectangles vers le bas, puis à droite. Le lecteur se rappellera que le nombre de formes possibles pour les motifs primitifs est le nombre d'ordonnancements des deux dimensions de l'image ; il retrouvera ici cette propriété.



(a) Patchs terminaux (b) Cellules principales (c) Extension à droite... (d) ...puis en bas.

FIGURE 4.34 – Transformation des patchs en motifs primitifs.

Vers la reconnaissance de langages

Notre algorithme peut faire davantage qu'une élection de classe d'équivalence (ou qu'une construction de motifs primitifs). En effet, une fois la configuration découpée en motifs primitifs, il est possible de simuler dans ces motifs les calculs de n'importe quel automate à entrée bornée agissant sur ces motifs primitifs, de la même manière que le calcul d'automates cellulaires à entrée bornée a été simulé dans les intervalles en dimension 1.

Cette simulation d'automates à entrée bornée nous amène à la reconnaissance de langages bi-dimensionnels. Toutefois, il n'est pas possible de reconnaître n'importe quel langage. De la même manière que notre algorithme en dimension 1 ne permettait la reconnaissance que des langages cycliques, notre algorithme en dimension 2 n'autorise que la reconnaissance de *langages de racines primitives*. Cette notion est délicate : étant donné qu'on ne sait pas décider si une image finie peut être la racine primitive d'une configuration donnée, il est difficile de savoir à quel point cette restriction est contraignante.

Toutefois, il existe des problèmes non triviaux que notre algorithme peut résoudre : par exemple, une version bi-dimensionnelle du problème de classification de densité [27], en d'autres termes, déterminer si une configuration périodique sur un alphabet $\{0, 1\}$ contient plus de 0 que de 1. Il est aussi possible de résoudre des problèmes portant sur le nombre de classes d'équivalence d'une configuration donnée, par exemple savoir si elle contient plus de classes d'équivalence que la taille de l'alphabet qu'elle utilise.

Chapitre 5

Élection de leader et motifs primitifs en dimension quelconque

Dans quelle mesure les résultats présentés aux chapitres précédents pour les configurations périodiques de dimension 1 ou 2 peuvent-ils se généraliser aux dimensions supérieures ? Cette question naturelle est l'objet de ce chapitre.

Tout d'abord, nous constaterons que la notion de racine primitive (ou motif primitif), vue au Chapitre 4, capable de paver une configuration périodique par translation uniforme, s'adapte à toute dimension d et nous donnerons une version affaiblie de la caractérisation de leurs dimensions possibles : les racines primitives d'une configuration d-périodique donnée peuvent avoir jusqu'à $d!$ dimensions distinctes $n_1 \times \ldots \times n_d$.

Le résultat principal du chapitre est une généralisation des algorithmes vus en dimensions 1 et 2 à toute dimension d : nous décrivons un automate cellulaire qui, pour toute configuration d-périodique, réalise l'élection d'un leader (classe d'équivalence de cellules) et en tire une partition de la configuration initiale par les translatés d'un certain motif primitif.

L'algorithme que nous présentons ici est, d'une certaine façon, une version simplifiée/épurée de l'algorithme étudié au chapitre précédent pour la dimension 2. Cet algorithme utilisait la notion d' « arbre de recouvrement » d'un patch. En dimension d, il s'agit de remplacer la notion d'intervalle en dimension 1 (cf. chapitre 3) et celle de patch en dimension 2 (chapitre 4) par une notion simple d'arbre, appelé d-arbre.

Cette « simplification » de l'algorithme nous permet de le généraliser à toutes les dimensions de manière uniforme. Par contre, nous verrons que cette simplicité se paie par une complexité en temps plus élevée même si ce temps reste polynomial en la taille d'un motif primitif.

5.1 Motifs primitifs

L'intégralité des définitions portant sur les motifs primitifs que nous avons présentées au chapitre précédent peuvent se généraliser aux images périodiques

de dimension quelconque. Nous allons les énoncer pour mémoire.

Définition 5.1 (image d-périodique)**.** *Soit Σ un alphabet fini, une* image d-périodique sur Σ *est une fonction $P : \mathbb{Z}^d \to \Sigma$ telle qu'il existe une famille libre de vecteurs $v_1, v_2, \ldots, v_d \in \mathbb{Z}^d$ telle que :*

$$\forall x \in \mathbb{Z}^d \ \forall i \in [\![1; d]\!] : P(x + v_i) = P(x)$$

Dans le contexte des images d-périodiques, un élément de $\mathbb{Z}^d$ est appelé voxel.

Définition 5.2 (fonctions shift)**.** *La famille de fonctions* shift $(\sigma_i)_{i \in [\![1;d]\!]}$ *est définie sur les images de la façon suivante :*

$$\forall x = (x_1, x_2, \ldots x_d) \in \mathbb{Z}^d : \sigma_i(P)(x) = P(x_1, x_2, \ldots, x_i + 1, \ldots, x_d)$$

Définition 5.3 (voxels équivalents)**.** *Deux voxels $x, y \in \mathbb{Z}^d$ d'une image P sont dits* équivalents *si la composition des fonctions shift qui amène x sur y laisse l'image inchangée, c'est à dire si $\sigma_1^{y_1 - x_1} \circ \sigma_2^{y_2 - x_2} \circ \ldots \sigma_d^{y_d - x_d}(P) = P$. On note alors $x \sim y$.*

Cette définition est encore une fois celle d'une relation d'équivalence. La remarque suivante est vérifiée, comme dans le cas de la dimension 2 :

Fait 5.4. *Pour toute image d-périodique P, il existe un nombre fini de classes d'équivalence de ses voxels. De plus, chaque classe d'équivalence contient un nombre infini de voxels. Enfin, la classe d'équivalence du voxel 0 constitue un* réseau entier de dimension d*, c'est à dire un sous-groupe de $(\mathbb{Z}^d, +)$.*

Définition 5.5 (motifs hyper-rectangulaires)**.** *Soit P une image d-périodique sur Σ, le* motif hyper-rectangulaire *de taille $n_1 \times n_2 \times \ldots n_d$ extrait en $x_0 \in \mathbb{Z}^d$ est la fonction suivante :*

$$\begin{array}{rccl} R_{x_0} : & [\![0, n_1 - 1]\!] \times [\![0, n_2 - 1]\!] \times \ldots \times [\![0, n_d - 1]\!] & \to & \Sigma \\ & x & \mapsto & P(x + x_0) \end{array}$$

Définition 5.6 (racines primitives)**.** *Un motif hyper-rectangulaire R_{x_0} de taille $n_1 \times \ldots \times n_d$ est une* racine primitive *de l'image P s'il contient exactement un représentant de chaque classe d'équivalence de P, c'est à dire si :*
$\forall x \in \mathbb{Z}^d$; $\exists! x' \in [\![0, n_1 - 1]\!] \times \ldots \times [\![0, n_d - 1]\!]$ tel que $x \sim x_0 + x'$.

5.1.1 Discussions sur les propriétés des racines primitives

Comme on l'a vu au cours du chapitre précédent, de nombreuses propriétés des racines primitives en dimension 2 se retrouvent en dimension quelconque. Nous allons rapidement revenir sur ces propriétés.

Existence des racines primitives

Le théorème 4.8, qui établit que toute image bi-périodique contient des racines primitives, peut être étendu en dimension quelconque. Son équivalent est alors :

Théorème 5.7. *Soit P une image d-périodique, alors il existe $x_0 \in \mathbb{Z}^d$ et $(n_1, \ldots, n_d) \in \mathbb{N}^d$ tels que le motif hyper-rectangulaire R_{x_0} de taille $n_1 \times \ldots \times n_d$ est une racine primitive de P.*

La preuve de ce théorème ne sera pas détaillée ici : elle utilise les mêmes étapes et les mêmes arguments que son homologue en dimension 2 (voir section 4.1.1) ; c'est à dire les propriétés des réseaux entiers et de la forme normale de Hermite des bases de ces réseaux entiers (voir [12]).

Caractérisation des racines primitives

Nous ne sommes pas parvenu à caractériser parfaitement les racines primitives des images d-périodiques de dimension quelconque. En effet, nous ne pouvons prouver que le théorème suivant, qui est une version affaiblie du théorème 4.10 :

Théorème 5.8. *Soit P une image d-périodique, et soit $S_P \subset \mathbb{N}^d$ l'ensemble de toutes les dimensions possibles des racines primitives de P (formellement, $S_P = \{(n_1, \ldots, n_d);\ \exists R_{x_0}$ une racine primitive de P de taille $n_1 \times \ldots \times n_d\}$), alors une racine primitive peut être extraite de n'importe quel point de l'image P, pourvu qu'elle soit de dimension appropriée : $\forall (n_1, \ldots, n_d) \in S_P;\ \forall x \in \mathbb{Z}^d$, si R_x est le motif rectangulaire de taille $n_1 \times \ldots \times n_d$ extrait de P en x, alors R_x est une racine primitive de P.*

De plus, les racines primitives d'une image peuvent avoir jusqu'à $d!$ dimensions différentes : $\exists P_0$ tel que $|S_{P_0}| = d!$;

En d'autres termes, nous savons qu'il peut exister $d!$ « formes » différentes possibles pour les motifs primitifs d'une image, mais nous ne savons pas s'il peut en exister davantage (contrairement au cas de la dimension 2, où nous savons que 2 est une borne maximale pour le nombre de formes des racines primitives d'une image).

L'affaiblissement de ce point du théorème est dû à une remarque géométrique simple que nous avons pu faire en dimension 2, et dont nous ignorons pour l'instant la validité en dimension supérieure. Cette remarque concerne les propriétés d'un pavage uniforme du plan par un rectangle :

Fait 5.9 (pavage uniforme en dimension 2)**.** *Soit un rectangle $V = [\![0, m-1]\!] \times [\![0, n-1]\!]$ et $\mathcal{L}$ un sous-groupe de $(\mathbb{Z}^2, +)$ tel que*

$$\forall z \in \mathbb{Z}^2\ \exists!\ v_z \in \mathcal{L} \text{ tel que } z \in v_z + V$$

(en d'autres termes, $\mathcal{L}$ est un pavage uniforme du plan par V),
alors on a :

$$\text{soit } \forall z = (x, y) \in \mathbb{Z}^2\ x = 0 \implies v_z \text{ est un vecteur vertical}$$

$$\text{soit } \forall z = (x, y) \in \mathbb{Z}^2\ y = 0 \implies v_z \text{ est un vecteur horizontal}$$

Cette remarque peut s'interpréter intuitivement de la manière suivante : dans tout pavage uniforme du plan, il existe une « ligne de fracture » soit horizontale, soit verticale (voir figure 5.1). L'analogue de cette remarque en dimension quelconque serait que pour tout pavage uniforme de $\mathbb{Z}^d$ par un hyper-rectangle, il existe un « hyperplan de fracture » selon une des d dimensions. Ce fait se formulerait par l'hypothèse suivante :

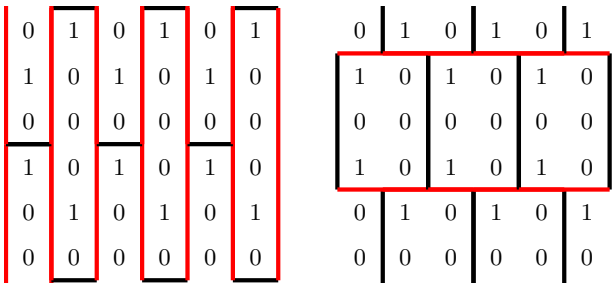
Conjecture 5.10. *Soit un hyper-rectangle* $V = [\![0, n_1 - 1]\!] \times \ldots \times [\![0, n_d - 1]\!]$ *et* $\mathcal{L}$ *un sous-groupe de* $(\mathbb{Z}^d, +)$ *tel que*

$$\forall z \in \mathbb{Z}^d \ \exists!\ v_z \in \mathcal{L} \text{ tel que } z \in v_z + V$$

alors $\exists i \in [\![1, d]\!]$ *tel que*
$\forall z = (z_1, \ldots, z_n) \in \mathbb{Z}^d \ z_i = 0 \implies (v_z)_i = 0$

Nous sommes convaincu que cette conjecture est vraie pour $d = 3$: il semble possible de la prouver « à la main » grâce à une étude de cas multiples. Cependant, cette preuve ne serait ni élégante, ni particulièrement utile, ni adaptable aux dimensions supérieures. C'est pour ces raisons qu'elle ne sera pas présentée.

Cette conjecture est de nature purement géométrique, et peut être énoncée de différentes manières. Étant donné qu'elle sort du cadre général de cette thèse, nous n'avons pas cherché à l'approfondir. Il est cependant probable qu'elle ait déjà été considérée, voire même résolue par des géomètres. Si tel est le cas, nous n'en sommes toutefois pas avertis. La preuve de cette conjecture serait suffisante à renforcer le théorème précédent ; il serait alors l'analogue parfait du théorème 4.10 en dimension quelconque.



(a) Fracture verticale (b) Fracture horizontale

FIGURE 5.1 – Mise en évidence des lignes de fracture d'un pavage uniforme en dimension 2.

5.2 Algorithme généralisé

Nous allons présenter dans cette section un algorithme d'élection de leader en dimension quelconque, qui permettra d'élire une unique classe d'équivalence parmi les voxels d'une configuration périodique de dimension d. De plus, l'élection de cette classe d'équivalence permettra de partitionner la configuration initiale de l'automate cellulaire en racines primitives. Cet algorithme est une généralisation de celui que nous avons présenté au chapitre précédent ; le lecteur retrouvera des notions familières, et les preuves seront similaires.

Problème de l'élection de leader : Le problème de l'élection de leader en dimension quelconque peut s'énoncer ainsi : il s'agit de construire un automate cellulaire agissant sur les configurations d-périodiques, de telle sorte qu'à partir d'un certain temps :

— Les états des cellules d'une unique classe d'équivalence appartiendront à un certain sous-ensemble d'états finaux $F \subseteq \mathcal{Q}$ et ne le quitteront plus.
— Les états de toutes les autres cellules resteront dans $\mathcal{Q} \setminus F$.

Théorème 5.11 (B.). *L'algorithme présenté dans cette section résout le problème de l'élection de leader en dimension quelconque en un temps $T(N)$ polynomial en le nombre N de classes d'équivalence de voxels de la configuration initiale.*

5.2.1 Présentation des outils

Les automates que nous considérerons dans cette section seront des automates cellulaires d-périodiques, c'est-à-dire des automates pour lesquels le réseau sous-jacent est $\mathbb{Z}^d$. De plus, ces automates vont tous utiliser le voisinage de Moore, c'est-à-dire $\mathcal{V} = \{v \in \mathbb{Z}^d;\ \max(|v_i|) \leq 1\}$.

Nous introduisons maintenant l'analogue en dimension quelconque des intervalles en dimension 1 et des patchs en dimension 2.

Définition 5.12 (d-arbre). *On appelle* d-arbre *un sous-graphe fini, connexe sans cycle du graphe de Cayley induit par $(\mathbb{Z}^d, +)$. La* taille *d'un d-arbre est son nombre de nœuds, noté a.*

La figure 5.2 donne un exemple de 3-arbre. Il est clair que la structure d'un d-arbre possédant a nœuds peut être codé dans les a cellules correspondantes d'un automate cellulaire de dimension d. Par la suite, lorsqu'on fera référence à un d-arbre, on supposera qu'il est codé dans un automate cellulaire de dimension d.

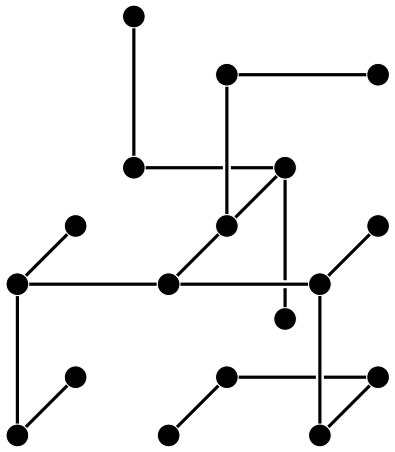


FIGURE 5.2 – Un exemple de 3-arbre

De même que nous avions défini le périmètre d'un patch comme le temps nécessaire à un signal pour parcourir sa frontière, nous définissons le périmètre d'un arbre de la manière suivante :

Définition 5.13 (périmètre). *On appelle* périmètre *d'un d-arbre A possédant a nœuds la valeur $n = 2da$.*

Cette définition trouvera sa justification dans la section suivante.

La notion d'adjacence entre les patchs en dimension 2 se traduisait par la notion de voisinage local. Les définitions suivantes vont nous permettre de définir, de manière similaire, une notion d'adjacence entre les d-arbres.

Définition 5.14 (cellules voisines)**.** *Soient $C_1, C_2 \in \mathbb{Z}^d$ deux cellules d'un automate cellulaire de dimension d, soit $A = (V, E)$ un d-arbre encodé dans l'automate cellulaire. C_2 est une* voisine *de C_1 dans A si :*

— $C_1 \in V$,
— $(C_1, C_2) \notin E$,
— $||(C_1, C_2)||_1 = 1$.

La figure 5.3 indique quelles sont les voisines d'une cellule particulière dans un 2-arbre. On note qu'une cellule d'un d-arbre peut avoir une voisine dans ce même arbre.

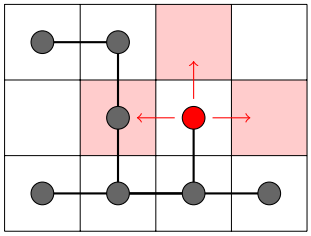


FIGURE 5.3 – Les voisines d'une cellule dans un 2-arbre.

Définition 5.15 (multi-ensemble des voisins)**.** *Le* multi-ensemble des voisins *d'un d-arbre A est le multi-ensemble résultant de la réunion disjointe des ensembles des cellules voisines des cellules de A.*

La multiplicité d'un élément dans cet ensemble est égale au nombre de cellules de l'arbre dont l'élément est une cellule voisine. La figure 5.4 donne le multi-ensemble des voisins de l'arbre donné en exemple.

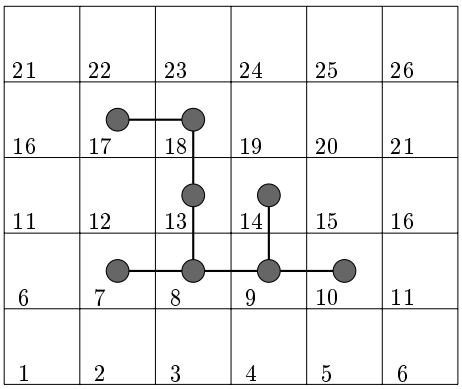


FIGURE 5.4 – Le multi-ensemble des cellules voisines de l'arbre est :
$\{2, 3, 4, 5, 11, 15, 15, 19, 13, 14, 19, 23, 22, 16, 12, 12, 12, 6\}$

Nous pouvons maintenant formaliser la notion intuitive d'adjacence entre deux d-arbres plongés dans $\mathbb{Z}^d$.

Définition 5.16 (adjacence d'arbres)**.** *Deux d-arbres disjoints $T_1 = (V_1, E_1)$ et $T_2 = (V_2, E_2)$ sont* adjacents *s'il existe $C_1 \in V_1$ et $C_2 \in V_2$ tels que C_2 est une cellule voisine de C_1 dans T_1 et C_1 est une cellule voisine de C_2 dans T_2.*

Nous voulons que notre algorithme puisse comparer deux arbres entre eux ; pour cela, nous introduisons un représentant canonique pour les d-arbres, sous la forme d'un mot unidimensionnel qui va encoder leur forme et leur contenu.

Définition 5.17 (mot de d-arbre)**.** *Soient $\mathcal{A}$ un automate cellulaire d-dimensionnel, et Σ son alphabet d'entrée. Soit A un d-arbre sur cet automate. On introduit un alphabet Λ tel que $|\Lambda| = 2d$ qui correspond aux $2d$ directions que peut prendre un signal se déplaçant sur le graphe de Cayley de $(\mathbb{Z}^d, +)$.*

Le mot de contour *de A est le mot $w_\Lambda \in \Lambda^*$ donnant la liste des déplacements qu'effectue le signal de A lors de son cycle (on s'intéresse uniquement aux déplacements, pas aux observations)*

Le mot de contenu *de A est le mot $w_\Sigma \in \Sigma^*$ donnant la liste des caractères d'entrée des cellules de A dans l'ordre d'un parcours en ordre préfixe de A.*

Finalement, le mot d'arbre *de A est le mot $w_A = w_\Lambda w_\Sigma$ résultant de la concaténation des mots de contour et de contenu de A.*

On dit que deux arbres A_1 et A_2 sont *différents* si leurs mots d'arbre sont différents. Il est aisé de remarquer que si deux mots d'arbre sont différents, alors ils sont également différents à une permutation circulaire près.

La figure 5.5 donne le mot d-arbre pour notre exemple.

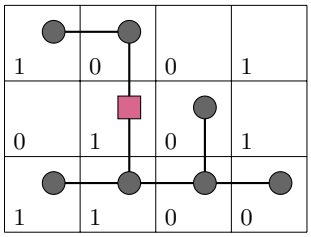


FIGURE 5.5 – Le mot de cet arbre est
$w_A = \uparrow\leftarrow\rightarrow\downarrow\downarrow\leftarrow\rightarrow\rightarrow\rightarrow\leftarrow\uparrow\downarrow\leftarrow\uparrow$ 10111000

5.2.2 Principe général de l'algorithme

L'algorithme généralisé utilise le même principe que les algorithmes en dimension 1 et 2 : Les cellules de l'automate cellulaire sont d'abord isolées les unes des autres, puis elles se regroupent au cours du temps. Dans chaque groupement, on identifie une cellule particulière. Lorsque les groupements de cellules ne peuvent plus croître, l'ensemble des cellules sélectionnées (une par groupement) forme la classe d'équivalence qui est élue. De la même manière que précédemment, on s'assurera que deux groupement différents voisins l'un de l'autre finiront par fusionner. Les groupements en dimension 1 étaient appelés *intervalles* et les groupements en dimension 2 étaient appelés *patchs* ; en dimension quelconque les groupements seront les *d-arbres*.

Arbres et signaux

À chaque instant de l'algorithme, la configuration de l'automate cellulaire est partitionnée en d-arbres. À chaque arbre est associé un unique signal qui va servir à déclencher sa fusion. Le signal va visiter toutes les cellules de l'arbre, et « observer »les voisines de ces cellules dans cet arbre. Lorsque deux signaux s'observent mutuellement, leurs deux arbres respectifs fusionnent en un seul arbre par l'ajout d'une arête entre les cellules contenant leurs deux signaux.

Chaque arbre est enraciné. La position de la racine est définie récursivement de la manière suivante :

Définition 5.18 (racine d'un d-arbre)**.** *À l'initialisation, l'arbre ne contient qu'une seule cellule, cette cellule est la* racine*. Lorsque deux arbres fusionnent par l'ajout d'une arête entre deux cellules voisines C_1 et C_2, la* racine *du nouvel arbre est la cellule parmi C_1 et C_2 qui a les coordonnées lexicographiquement maximales.*

Notons que le changement de racine peut être détecté et traité algorithmiquement en envoyant un signal dans l'arbre lors d'une fusion. Ce signal va effacer les anciennes racines des deux arbres fusionnés. La figure 5.6 illustre le processus de fusion des arbres et de choix de la racine.

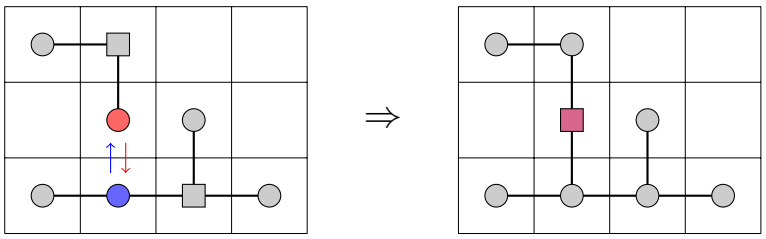


FIGURE 5.6 – La fusion de deux 2-arbres. Les nœuds carrés symbolisent la racine des arbres, les couleurs symbolisent la position des signaux.

Trajet du signal

Lorsqu'un nouvel arbre est créé, sa racine génère le signal qui va le parcourir. Le signal se comporte comme en dimension 2, c'est à dire qu'il suit un chemin qui forme une boucle, en marquant parfois des temps d'arrêt. Le signal effectue un parcours en profondeur du graphe de Cayley induit par $(\mathbb{Z}^d, +)$, à ceci près que son chemin est « coupé » lorsqu'il doit traverser une arête qui n'appartient pas à son d-arbre. Quand son chemin est coupé, il « observe » la cellule dans laquelle il aurait dû se rendre pendant une étape de calcul, puis continue le parcours en profondeur jusqu'à retourner à la racine.

La figure 5.7 résume le trajet cyclique d'un signal partant de la racine. On remarque que le nombre d'« observations » qu'effectue le signal est exactement le cardinal du multi-ensemble des voisins de l'arbre.

Nous allons maintenant définir l'analogue du point d'attente principal d'un patch sur les arbres, la *cellule voisine principale*, afin de pouvoir induire un comportement particulier lorsque le signal attendra en observant cette voisine.

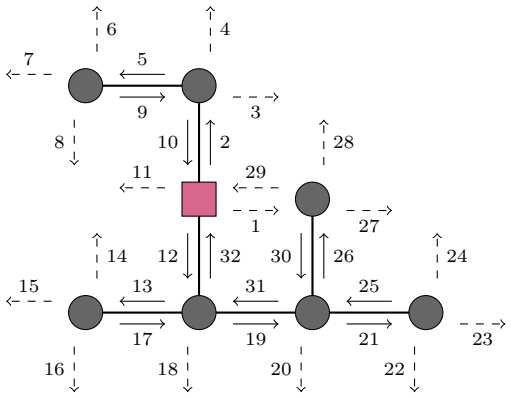


FIGURE 5.7 – Le voyage du signal sur un 2-arbre. Les flèches pleines indiquent les déplacements du signal, les flèches pointillées indiquent les observations.

Définition 5.19 (voisine principale d'un arbre)**.** *La* voisine principale *d'un d-arbre est la première cellule voisine observée par un signal partant de la racine du d-arbre.*

À propos du périmètre Intéressons nous maintenant au temps que met le signal pour effectuer un tour complet, s'il ne s'arrête pas. Le signal doit effectuer un parcours en profondeur de l'arbre et remonter jusqu'à la racine (il doit pour cela parcourir 2 fois chacune des $a-1$ arêtes de l'arbre), et observer exactement une fois chacune des cellules du multi-ensemble des voisins de l'arbre (chacune des a cellules de l'arbre a $2d$ voisins, auxquels il faut retirer 2 fois chacune des $a-1$ arêtes). Le signal effectue donc un cycle complet en un temps $(2da - 2(a-1)) + 2(a-1) = 2da$. On trouve ici la justification du terme de *périmètre* pour la valeur $n = 2da$.

On vérifie que le périmètre de l'arbre présenté dans la figure 5.7 est bien $32 = 2 \times 2 \times 8$.

Comportement du signal

Nous allons maintenant détailler le comportement du signal qui voyagera dans les d-arbres. Pour cela, on introduit quelques notations.

Notations 5.20 (k,τ_n,val)**.**
Pour un d-arbre de périmètre n,

- $k = 2d + |\Sigma|$ *est la taille de l'alphabet utilisé par les mots de d-arbre.*
- $\tau_n = kn^2$ *est le temps d'attente du signal sur chaque cellule voisine.*
- $val \ : \ \Lambda \cup \Sigma \to [\![0; k-1]\!]$ *est une bijection quelconque.*

De la même manière que dans les algorithmes en dimensions 1 et 2, chaque d-arbre A va produire à la demande les symboles du mot de d-arbre w_A qui lui correspond. Le i-ème symbole du mot sera noté $w_A(i)$. Le lecteur se convaincra facilement que la racine de l'arbre peut fournir ces symboles si un délai linéaire ($O(n)$) est respecté entre chaque requête. Nous allons voir plus tard que ce délai est effectivement respecté.

Lors de la création du d-arbre, le signal est créé par la racine, puis va se déplacer selon les modalités évoquées précédemment pour observer la voisine principale de l'arbre. Ensuite, le comportement du signal suit le cycle suivant :

- Observer la cellule voisine sur laquelle est positionnée le signal pendant un temps τ_n.
- Si le signal vient d'observer la voisine principale, effectuer $val(w_A(i))$ tours complets de l'arbre, puis ordonner à la racine de l'arbre de récupérer le symbole suivant. Si la racine contenait le dernier symbole du mot, récupérer le premier symbole à nouveau.
- Faire un tour complet de l'arbre, puis se positionner sur la cellule voisine suivante dans l'ordre de parcours.

À tout instant dans cet algorithme, si deux signaux s'observent mutuellement, ils sont supprimés et leurs arbres respectifs fusionnent, puis un nouveau signal est créé à la racine du nouvel arbre (*i.e.* l'un des deux points de fusion).

Remarque : Le temps d'attente du signal sur une cellule voisine est borné par $O(n^2)$. Étant donné que le signal doit attendre une fois sur chaque cellule voisine avant de retourner sur la voisine principale, et que le nombre de cellules voisines d'un d-arbre est de $O(n)$, nous pouvons établir que le symbole du mot de d-arbre stocké dans la racine doit changer avec un délai $O(n^3)$. Ce délai est suffisant pour que le signal n'ait pas à attendre à la racine qu'un nouveau symbole arrive lorsqu'il est nécessaire.

5.2.3 Preuve de l'algorithme

Afin de prouver le théorème 5.11, nous allons d'abord vérifier le lemme suivant :

Lemme 5.21. *Si deux d-arbres* différents A_1 *et* A_2 *de périmètres* n_1 *et* n_2 *sont adjacents à un instant* t*, alors au moins l'un d'entre eux doit avoir fusionné avant l'instant* $t + T(\max(n_1, n_2))$*, où* $T(n)$ *est une fonction polynomiale en* n*.*

Démonstration. La preuve du lemme ne sera pas complètement détaillée dans cette section. En effet, elle est extrêmement similaire à la fois dans son intuition et dans sa mise en œuvre à celle du chapitre précédent (voir la section 4.2.5). L'idée générale est la suivante : on considère deux d-arbres adjacents et différents A_1 et A_2 à un instant donné t, puis on prouve que s'ils n'ont pas fusionné avec un autre d-arbre avant, alors ils fusionneront ensemble au temps $t+T(\max(n_1, n_2))$, où $T(n)$ est une fonction polynomiale en n.

On peut supposer sans perdre de généralité que $n_1 \geq n_2$. Nous sommes amenés à distinguer deux sous-cas : le cas $n_1 > n_2$ et le cas $n_1 = n_2$.

Le cas $n_1 > n_2$: Comme précédemment, ce cas se prouve en établissant que pendant que le signal S_1 de l'arbre A_1 observe une cellule de l'arbre A_2 il est nécessairement lui-même observé par le signal S_2 de A_2, ce qui déclenche une fusion. La preuve se base une fois encore sur une inégalité du type :

$$k(n+1)^2 > kn^2 + n$$

Le cas $n_1 = n_2$: Ce cas est lui aussi traité avec les mêmes outils que précédemment : on observe les allers et venues des deux signaux sur un point d'adjacence quelconque entre les deux d-arbres (le diagramme de présence des signaux est similaire, voire identique à celui présenté sur la figure 4.33). À chaque instant où le signal S_1 s'arrête sur ce point d'adjacence et observe la cellule voisine pendant τ_n, nous associons une paire de symboles (a_i, b_j) des mots d'arbre de A_1 et A_2.

Étant donné que les mots d'arbre de A_1 et A_2 sont différents, nous établissons qu'il existe un instant pour lequel la paire (a_i, b_j) sera composée de symboles différents. On peut définir un décalage δ entre les signaux à cet instant particulier, et finalement remarquer que ce décalage sera « compensé » par la différence entre $val(a_i)$ et $val(b_j)$, ce qui contraindra les signaux de A_1 et A_2 à s'observer mutuellement.

La méthode utilisée pour prouver ce fait est légèrement différente selon le signe de $val(a_i) - val(b_j)$, comme précédemment, et utilise une fois encore des bornes fines sur le décalage δ entre les signaux.

Complexité temporelle

Nous avons déjà remarqué que le symbole du mot d'arbre contenu dans la racine changeait avec une période $O(n^3)$. De plus, il est évident que la longueur du mot d'un d-arbre de périmètre n est linéaire en n. Finalement, nous pouvons établir que la racine d'un d-arbre produit les symboles du mot de contenu au cours d'un cycle de longueur $O(n^4)$.

Nous avons établi précédemment que si deux d-arbres différents sont adjacents à un instant donné t, alors ils doivent fusionner avant que l'intégralité des symboles des mots de contenu de ces deux arbres aient été produits par leurs racines respectives, c'est-à-dire avant un temps $t + O(n^4)$, ce qui conclut la preuve du lemme 5.21. □

Du lemme 5.21 au théorème 5.11

Le lemme 5.21 nous garantit que tant qu'il existera deux d-arbres différents et adjacents dans la configuration de l'automate cellulaire d-périodique, alors ces d-arbres fusionneront. Nous savons également que le nombre de cellules d'un d-arbre ne peut pas excéder le nombre N, toujours défini comme le nombre de classes d'équivalence de la configuration initiale. De plus, nous savons que tant que la configuration de l'automate ne sera pas entièrement composée de translatés d'un seul d-arbre de taille maximale, il existera des d-arbres différents et adjacents dans cette configuration (de par la définition même des classes d'équivalence).

Nous pouvons par conséquent établir qu'il existe un instant $T(N)$ à partir duquel la configuration de l'automate cellulaire sera entièrement partitionnée en d-arbres identiques de taille maximale, et que ces arbres ne fusionneront plus à partir de l'instant $T(N)$. Nous allons maintenant donner une borne à ce $T(N)$.

À aucun instant de l'algorithme la taille d'un d-arbre ne saurait excéder N. D'autre part, le périmètre d'un arbre est linéaire en sa taille, le périmètre d'un d-arbre au cours de l'algorithme appartient donc à $O(N)$.

Considérons les N cellules dont sera composé un d-arbre de taille maximale à la « fin » de l'algorithme. Considérons l'ensemble $\mathcal{E}$ des d-arbres formés par ces cellules ; à l'initialisation $\mathcal{E}$ est de cardinal N, puisque chaque cellule forme un

d-arbre indépendant. Le cardinal de $\mathcal{E}$ va diminuer au cours du temps, jusqu'à atteindre 1 (lorsque toutes les cellules seront réunies dans un d-arbre de taille maximale). On sait que tant que $|\mathcal{E}| > 1$ il existe dans $\mathcal{E}$ des d-arbres différents adjacents. Par conséquent, le cardinal de $\mathcal{E}$ doit diminuer strictement jusqu'à 1. Le temps entre deux décrémentations successives est bornée par $O(N^4)$.

Finalement, nous pouvons établir que $T(N) \in O(N^5)$. Nous définissons le sous-ensemble d'états finaux $F \subseteq \mathcal{Q}$ comme l'ensemble des états que peut prendre une cellule en étant la racine d'un d-arbre. En gardant cette définition à l'esprit, nous pouvons conclure que notre algorithme résout effectivement le problème de l'élection de leader multi-dimensionnelle en temps polynomial. La classe d'équivalence qui est élue est la classe des cellules racines des d-arbres de taille maximale.

De l'élection de leader à la partition en racines primitives

Il existe un instant dans la vie d'un d-arbre où cet arbre est assuré que les arbres qui lui sont adjacents sont identiques à lui-même. Cet instant peut être détecté dès que la racine a produit tous les symboles du mot d'arbre et que tous les arbres adjacents ont eu un comportement « normal » (s'ils n'ont pas fusionné). Passé cet instant, il est possible de construire localement un pavage de l'espace d-dimensionnel par ce qui serait un motif primitif si le calcul était effectivement terminé (rappelons qu'il est impossible de se rendre compte localement que le calcul global est terminé).

La procédure pour construire le pavage par racine primitive est la suivante :

— Fixer un ordre arbitraire sur les d dimensions et considérer l'hyper-rectangle constitué de la cellule contenant la racine de l'arbre.

— Pour chaque dimension, étendre l'hyper-rectangle jusqu'à ce qu'il soit bloqué par l'hyper-rectangle d'un autre d-arbre.

— Lorsque l'hyper-rectangle a été étendu selon les d dimensions, et si le calcul est effectivement terminé, la configuration est alors partitionnée en racines primitives.

Remarques : On retrouve dans les choix arbitraires de l'ordre des d dimensions les $d!$ formes potentielles pour les racines primitives, que nous avions évoquées dans la première section de ce chapitre. À tout instant, si l'hyper-rectangle en extension rencontre un d-arbre en train de fusionner, il s'efface (cela signifie en effet que le calcul n'était pas globalement terminé). Une illustration de cette procédure en dimension 2 peut être observée sur la figure 4.34.

5.2.4 Remarques finales

Le lecteur attentif aura remarqué que l'algorithme présenté dans ce chapitre est plus simple que son homologue spécifique à la dimension 2 : les problématiques de « patch propre », de synchronisation et d'élection de leader sur la frontière, qui se posaient en dimension 2, n'ont pas lieu d'être lorsque les patchs sont remplacés par des d-arbres.

Toutefois, le travail qui a été effectué sur l'algorithme en dimension 2 n'a pas été vain : rappelons nous que notre paramètre de complexité en dimension 2 était le nombre de points d'attente de la *frontière du patch*, et que les patchs que nous traitions étaient propres, c'est-à-dire que leur frontière était d'une certaine

façon minimale. Dans l'algorithme décrit ici, les signaux de fusion parcourent systématiquement l'intégralité de l'arbre, et observent souvent des cellules voisines qui font en réalité partie du même d-arbre. À l'exception de quelques cas pathologiques (voir figure 4.22), ces observations superflues étaient évitées en dimension 2. De plus, même si dans le pire des cas la longueur de la frontière d'un patch est linéaire en son nombre de cellules, typiquement elle est de l'ordre de la racine carrée de ce nombre de cellules. Cette propriété ne s'applique pas aux d-arbres, dont l'intérieur est vide.

Même si les bornes de complexité grossières sont les mêmes dans les deux cas (c'est-à-dire $O(N^5)$), le lecteur pourra se convaincre que l'algorithme dédié à la dimension 2 est plus efficace en pratique, d'autant plus que nombre des traitements des patchs peuvent être parallélisés.

Chapitre 6

Conclusion et perspectives

Les travaux présentés dans cette thèse ont été motivés par des questions portant sur la notion naturelle de « calcul périodique uniforme ». Que se passe-t-il lorsqu'un modèle de calcul ne dispose d'aucun point de repère, ni dans son espace de travail (comme la tête d'une machine de Turing, ou son équivalent) ni sur son entrée (comme dans le cas des automates cellulaires à entrée bornée) ? Dans de telles conditions, que peut-on appeler un calcul ? Comment détecter sa terminaison ou définir son résultat ?

Nous sommes parvenus à des réponses satisfaisantes et relativement complètes dans le cadre des automates cellulaires de dimension 1, qui sont le modèle de calcul uniforme par excellence. Nous nous sommes rendus compte que la plupart des concepts utilisés dans le cadre de la dimension 1 s'étendaient naturellement aux automates cellulaires de dimensions supérieures. Nous avons par conséquent tenté d'étendre les résultats obtenus, en particulier dans le cas de la dimension 2.

Les difficultés que nous avons rencontrées nous ont permis de comprendre que le calcul uniforme en dimension quelconque comporte (au moins) deux aspects intéressants : la nature des objets manipulés (les racines primitives) et les techniques à utiliser pour les construire (les algorithmes d'élection de leader). C'est autour de ces deux aspects que s'est articulé notre travail.

Naturellement, il a été impossible de traiter toutes les questions ouvertes survenues lors de nos travaux. On se propose ici de synthétiser celles qui nous ont semblé les plus pertinentes, et d'ouvrir quelques pistes de réflexion.

6.1 Racines primitives en dimensions supérieures

Comme nous l'avons vu, la notion de racine primitive d'une image d-périodique de dimension quelconque est naturelle et s'exprime facilement. Pourtant, nous n'avons pas été en mesure d'en apporter une caractérisation exacte dans le cas $d > 2$. En effet, la question de savoir s'il peut exister plus que $d!$ formes possibles pour les racines primitives d'une image d-périodique donnée reste ouverte. Rappelons nous que nous n'avons pas su répondre à cette question car nous sommes incapables de prouver notre hypothèse 5.10 (voir page 82). Nous sommes toutefois convaincu que cette hypothèse n'est pas intrinsèquement difficile, et que notre incapacité à la prouver est due à notre manque d'expertise

dans le domaine de la géométrie discrète en dimension quelconque.

Question 1. *Existe-t-il des images d-périodiques dont les racines primitives ont plus de $d!$ formes différentes ?*

Nous avons étudié en détail l'extraction de racines primitives d'une image d-périodique. Cependant, le problème inverse est également intéressant : à partir d'un motif rectangulaire fini de dimension d, est-il possible de construire une image d-périodique telle que le motif fini soit l'une de ses racines primitives ? La réponse à cette question dans le cas général est non : nous avons vu que toute racine primitive est nécessairement un mot primitif dans l'une de ses dimensions. Toutefois, on peut se demander si cette condition est suffisante, autrement dit si :

Question 2. *Étant donné un motif hyper-rectangulaire R de dimension d, tel que R est un mot primitif selon l'une de ses dimensions, existe-t-il une image d-périodique I telle que R soit une racine primitive de I ?*

Une réponse positive à cette question nous permettrait de caractériser des *langages de racines primitives*, qui seraient l'analogue en dimensions supérieures des langages cycliques. Nous serions alors en mesure d'établir des résultats d'équivalence entre la reconnaissance des langages de racines primitives sur les automates cellulaires à entrée bornée et les automates cellulaires à entrée périodique, grâce à des techniques similaires à celles utilisées dans le cas de la dimension 1. De plus, l'étude de tels langages pourrait être intéressante : le langage des mots primitifs est en lui-même à l'heure actuelle un objet d'étude [16]. *A fortiori*, il est raisonnable de supposer que sa généralisation en dimensions supérieures présente elle aussi des propriétés dignes d'intérêt.

6.2 Vers la reconnaissance forte en dimensions supérieures

Si on souhaitait parler de reconnaissance de langage sur les automates cellulaires d-périodiques (ce qui impliquerait que l'on sache caractériser un langage de racines primitives), il serait naturel de définir, de manière analogue à la dimension 1, une notion de reconnaissance faible et une notion de reconnaissance forte. Un langage de racines primitives en dimension d serait alors *faiblement reconnaissable* s'il existait un automate cellulaire tel que toute configuration générée à partir d'un élément de ce langage évolue jusqu'à atteindre un cycle limite dans lequel l'acceptation ou le rejet du mot puisse être lu sur n'importe quelle cellule de la configuration. Un langage serait *fortement reconnaissable* si le cycle limite atteint par l'automate est en fait un point fixe (cycle de longueur 1).

Nous avons prouvé que les notions de reconnaissance faible et forte sont équivalentes en dimension 1, et ce, de manière constructive. Il est légitime de se demander si cette équivalence est également vraie en dimension supérieure.

Question 3. *Tout langage de racines primitives de dimension d faiblement reconnaissable est-il fortement reconnaissable ?*

6.3 La complexité de l'élection de leader

Nous avons présenté des algorithmes résolvant le problème d'élection de leader des configurations périodiques dans différentes dimensions, notamment un algorithme résolvant le problème en dimension quelconque. Nous avons prouvé que ces algorithmes s'exécutaient en un temps polynomial en le nombre N de classes d'équivalence des cellules de la configuration initiale. Cependant, il est clair que la finesse de l'analyse de complexité n'a pas été notre principal objectif : des bornes polynomiales grossières ont été utilisées, et l'aspect massivement parallèle du modèle de calcul n'a pas été pleinement exploité. Nous sommes convaincu qu'une analyse fine des algorithmes nous permettrait d'abaisser la borne de complexité de $O(N^5)$ à $O(N^4)$. Cependant, la question de la complexité intrinsèque de notre problème n'a absolument pas été traitée.

Question 4. *Quelle est la complexité précise du problème de l'élection de leader périodique ? En particulier, ce problème est-il linéaire ?*

Il nous est apparu que le problème de l'élection de leader périodique est un problème de base du calcul périodique uniforme : il permet en effet de « briser » au maximum le caractère périodique de l'entrée et de dresser un pont entre le calcul sur entrée périodique et le calcul sur entrée bornée. Dès lors, le fait de déterminer sa complexité exacte permettrait de disposer d'un problème de référence dans le cadre d'une « théorie de la complexité périodique uniforme ».

6.4 Algorithmique en dimensions supérieures

Au cours de cette thèse, nous avons développé des outils algorithmiques propres aux dimensions supérieures à 1. Nous avons notamment généralisé la notion de signal de manière non triviale : en plus d'un ensemble d'états se déplaçant sur la configuration, les signaux que nous utilisons ont un *support* sous la forme d'un d-arbre ou de la frontière d'un patch. Cette extension entre dans le cadre d'une problématique soulevée entre autres par [44], sur la généralisation en dimension quelconque du concept de signal.

Il est également intéressant de constater que le concept de signal encodant la longueur d'un intervalle a été introduite (à notre connaissance) dans un article traitant de la construction d'ensembles μ-limites [9], c'est-à-dire dans un contexte de systèmes dynamiques discrets. Nous avons modifié cet outil algorithmique pour concevoir un signal faisant office d'« horloge », dont les variations de battements encodent la forme et le contenu de son support. Ce type d'outil, qui transforme des différences spatiales en différences temporelles, pourrait avoir des applications algorithmiques dans d'autres domaines.

Liste des symboles

δ	La fonction de transition locale d'un automate cellulaire
$\mathcal{A}$	Un automate cellulaire
$\mathcal{C}$	La configuration d'un automate cellulaire
$\mathcal{L}$	Un langage sur Σ^*
$\mathcal{Q}$	L'ensemble des états d'un automate cellulaire
$\mathcal{V}$	Le voisinage d'un automate cellulaire
Σ	L'alphabet d'entrée d'un automate cellulaire
A	Un d-arbre de cellules
a	Le nombre de cellules dans un intervalle/patch/arbre
F_δ	La fonction de transition globale d'un automate cellulaire
I	Un intervalle de cellules (dimension 1)
N	La taille d'une configuration périodique
n	Le périmètre d'un intervalle/patch/arbre
P	Un patch de cellules (dimension 2)

Bibliographie

[1] Nicolas Bacquey. The packing problem : A divide and conquer algorithm on cellular automata. In *Automata & JAC 2012, Local proceedings*, pages 1–10, 2012.

[2] Nicolas Bacquey. Complexity classes on spatially periodic cellular automata. In Ernst W. Mayr and Natacha Portier, editors, *STACS*, volume 25 of *LIPIcs*, pages 112–124, 2014.

[3] Nicolas Bacquey. Leader election on two-dimensional periodic cellular automata. *currently submitted*, 2015.

[4] Nicolas Bacquey. Primitive roots of bi-periodic infinite pictures. In Dirk Nowotka Florin Manea, editor, *Combinatorics on Words, 10th International Conference WORDS 2015. Local Proceedings*, number 2015/5 in Kiel Computer Science Series, pages 1–16, 2015.

[5] Robert Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1) :22–42, 1967.

[6] Marie-Pierre Béal, Olivier Carton, and Christophe Reutenauer. Cyclic languages and strongly cyclic languages. In *STACS*, volume 1046 of *LCNS*, pages 49–59, Berlin, 1996. Springer.

[7] Danièle Beauquier and Maurice Nivat. On translating one polyomino to tile the plane. *Discrete & Computational Geometry*, 6(1) :575–592, 1991.

[8] W. T. Beyer. Recognition of topological invariants by iterative arrays. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1969.

[9] Laurent Boyer, Martin Delacourt, and Mathieu Sablik. Construction of μ-limit sets. *Proceedings of JAC 2010*, page 76, 2010.

[10] Arthur W. Burks, editor. *Essays on Cellular Automata*. University of Illinois Press, Urbana, 1970.

[11] Olivier Carton. A hierarchy of cyclic languages. *ITA*, 31(4) :355–369, 1997.

[12] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag Berlin Heidelberg, 1993.

[13] Marianne Delorme and Jacques Mazoyer. Algorithmic tools on cellular automata. In Rozenberg et al. [45], pages 77–122.

[14] Marianne Delorme and Jacques Mazoyer. *Cellular Automata : a parallel model*, volume 460. Springer Science & Business Media, 2013.

[15] Pál Dömösi, S Horváth, and M Ito. Formal languages and primitive words. *Publicationes Mathematicae Debrecen*, 42(3–4) :315–321, 1993.

[16] Pál Dömösi and Masami Ito. *Context-Free Languages and Primitive Words*. World Scientific, 2014.

[17] G Bard Ermentrout and Leah Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of theoretical Biology*, 160(1) :97–133, 1993.

[18] Nazim Fatès. Stochastic cellular automata solve the density classification problem with an arbitrary precision. In *STACS*, volume 9, pages 284–295, 2011.

[19] Henryk Fúks. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55 :R2081–R2084, 1997.

[20] Dora Giammarresi and Antonio Restivo. Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence*, 6(02n03) :241–256, 1992.

[21] Gustav A Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Theory of computing systems*, 3(4) :320–375, 1969.

[22] Chuzo Iwamoto, Tomonobu Hatsuyama, Kenichi Morita, and Katsunobu Imai. On time-constructible functions in one-dimensional cellular automata. In *Fundamentals of Computation Theory*, pages 316–326. Springer, 1999.

[23] Jarkko Kari. Reversibility and surjectivity problems of cellular automata. *Journal of Computer and System Sciences*, 48(1) :149–182, 1994.

[24] Jarkko Kari. Theory of cellular automata : A survey. *Theoretical Computer Science*, 334(1-3) :3–33, 2005.

[25] Jarkko Kari. Basic concepts of cellular automata. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 3–24. Springer, 2012.

[26] S. Rao Kosaraju. On some open problems in the theory of cellular automata. *IEEE Transactions on Computers*, C-23(6) :561–565, 1974.

[27] Mark WS Land and Richard K Belew. No two-state CA for density classification exists. *Physical Review Letters*, 74(25) :5148–5150, 1995.

[28] Hans Lang, Manfred Schimmler, Hartmut Schmeck, and H. Schröder. Systolic sorting on a mesh-connected network. *IEEE Transactions on Computers*, 34(7) :652–658, 1985.

[29] Christopher G. Langton. Self-reproduction in cellular automata. *Physica D : Nonlinear Phenomena*, 10 :135 – 144, 1984.

[30] Stefano Levialdi. On shrinking binary picture patterns. *Commun. ACM*, 15(1) :7–10, January 1972.

[31] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2) :183–238, 1987.

[32] Jacques Mazoyer. An overview of the firing squad synchronization problem. In *Automata Networks*, volume 316 of *Lecture Notes in Computer Science*, pages 82–94. Springer Berlin / Heidelberg, 1988.

[33] Jacques Mazoyer. Computations on one-dimensional cellular automata. *Annals of Mathematics and Artificial Intelligence*, 16(1) :285–309, 1996.

[34] Jacques Mazoyer and Véronique Terrier. Signals in one-dimensional cellular automata. *Theoretical Computer Science*, 217(1) :53–80, 1999.

[35] Jacques Mazoyer and Jean-Baptiste Yunès. Computations on cellular automata. In Rozenberg et al. [45], pages 159–188.

[36] John McCarthy and Marvin Minsky. *Sequential Machines : Selected Papers*, pages 213–214. Addison-Wesley Longman Ltd., Essex, UK, 1964.

[37] Edward F. Moore. Machine models of self-reproduction. In *Proceedings of Symposia in Applied Mathematics*, volume 14, pages 17 – 33, 1962.

[38] John Myhill. The converse of Moore's garden-of-eden theorem. In *Proceedings of the American Mathematical Society*, volume 14, pages 658 – 686, 1963.

[39] Codrin Nichitiu, Jacques Mazoyer, and Eric Rémila. Algorithms for leader election by cellular automata. *Journal of Algorithms*, 41(2) :302–329, 2001.

[40] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[41] H Petersen. On the language of primitive words. *Theoretical Computer Science*, 161(1) :141–156, 1996.

[42] Victor Poupet. *Automates cellulaires : temps réel et voisinages*. PhD thesis, École Normale Supérieure de Lyon, 2006.

[43] Victor Poupet. Cellular automata : Real-time equivalence between one-dimensional neighborhoods. *Theory of Computing Systems*, 40(4) :409–421, 2007.

[44] Gaétan Richard. *Système de particules et collisions dans les automates cellulaires*. PhD thesis, Aix Marseille Université, 2008.

[45] Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors. *Handbook of Natural Computing*, volume 1. Springer, 2012.

[46] Palash Sarkar. A brief history of cellular automata. *ACM Computing Surveys*, 32(1) :80 – 107, 2000.

[47] C. Schnorr and A. Shamir. An optimal sorting algorithm for mesh connected computers. In *STOC*, pages 255–263, 1986.

[48] Alvy R. Smith III. Cellular automata complexity trade-offs. *Information and Control*, 18(5) :466 – 482, 1971.

[49] Véronique Terrier. Two-dimensional cellular automata recognizer. *Theor. Comput. Sci.*, 218(2) :325–346, 1999.

[50] Véronique Terrier. *Language Recognition by Cellular Automata*, pages 124–158. Volume 1 of Rozenberg et al. [45], 2012.

[51] Tommaso Toffoli and Norman Margolus. *Cellular automata machines : a new environment for modeling*. MIT press, 1987.

[52] Hiroshi Umeo. Firing squad synchronization problem in cellular automata. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 3537–3574. Springer New York, 2009.

[53] John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[54] Hao Wang. Proving theorems by pattern recognition I. *Communications of the ACM*, 3(4) :220–234, 1960.

Automates Cellulaires : Aspects algorithmiques des configurations périodiques en toute dimension

Résumé : Cette thèse analyse les capacités de calcul des automates cellulaires travaillant sur les configurations périodiques de dimension quelconque. D'une part, nous étudions les objets maximaux identifiables par ces automates cellulaires ; nous appelons ces objets les racines primitives des configurations périodiques de dimension quelconque. Nous en présentons une caractérisation et mettons en évidence certaines de leurs propriétés.
D'autre part, nous présentons un ensemble d'algorithmes, chacun adapté à une ou plusieurs dimensions particulières, permettant aux automates cellulaires d'extraire les racines primitives des configurations périodiques sur lesquelles ils sont appliqués. Ceux-ci utilisent des outils algorithmiques originaux étendant la notion de signal sur les automates cellulaires en dimension quelconque.
Au-delà des aspects techniques et algorithmiques, cette thèse pose les bases du calcul périodique uniforme, c'est-à-dire du calcul effectué sur un modèle dans lequel le programme et l'entrée sont isotropes. Nous y abordons notamment les problématiques de l'arrêt d'un tel calcul, de lecture de son résultat et de sa complexité en temps et en espace.

Mots-clefs : automates cellulaires ; algorithmes parallèles ; complexité de calcul ; systèmes dynamiques ; fonctions calculables

Cellular Automata : Algorithmic behaviour of periodical configurations of any dimension

Abstract : This thesis analyses the computational capabilities of cellular automata working on periodical configurations of any dimension. We first study the maximal objects these cellular automata can identify ; we call those objects primitive roots of periodical configurations of any dimension. We characterize them and show some of their properties.
Secondly, we present a set of algorithms on cellular automata, each one adapted to one or more dimensions, that extract primitive roots from the periodical configurations on which they are applied. Those algorithms use original tools that extend the notion of signals on cellular automata.
Beyond its technical and algorithmical aspects, this thesis lays the foundations of uniform periodical computation, *i.e.* computation performed on a model whose program and entry data are isotropic. In particular, we address the issues of halting such computation, reading its result and defining its temporal or spatial complexity.

Keywords : cellular automata ; parallel algorithms ; computational complexity ; dynamical systems ; computable functions

Discipline : Informatique et applications
Laboratoire : Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen (GREYC - UMR 6072)